Beyond Aspect-Oriented Programming: Toward Naturalistic Programming

Cristina Videira Lopes

Institute for Software Research

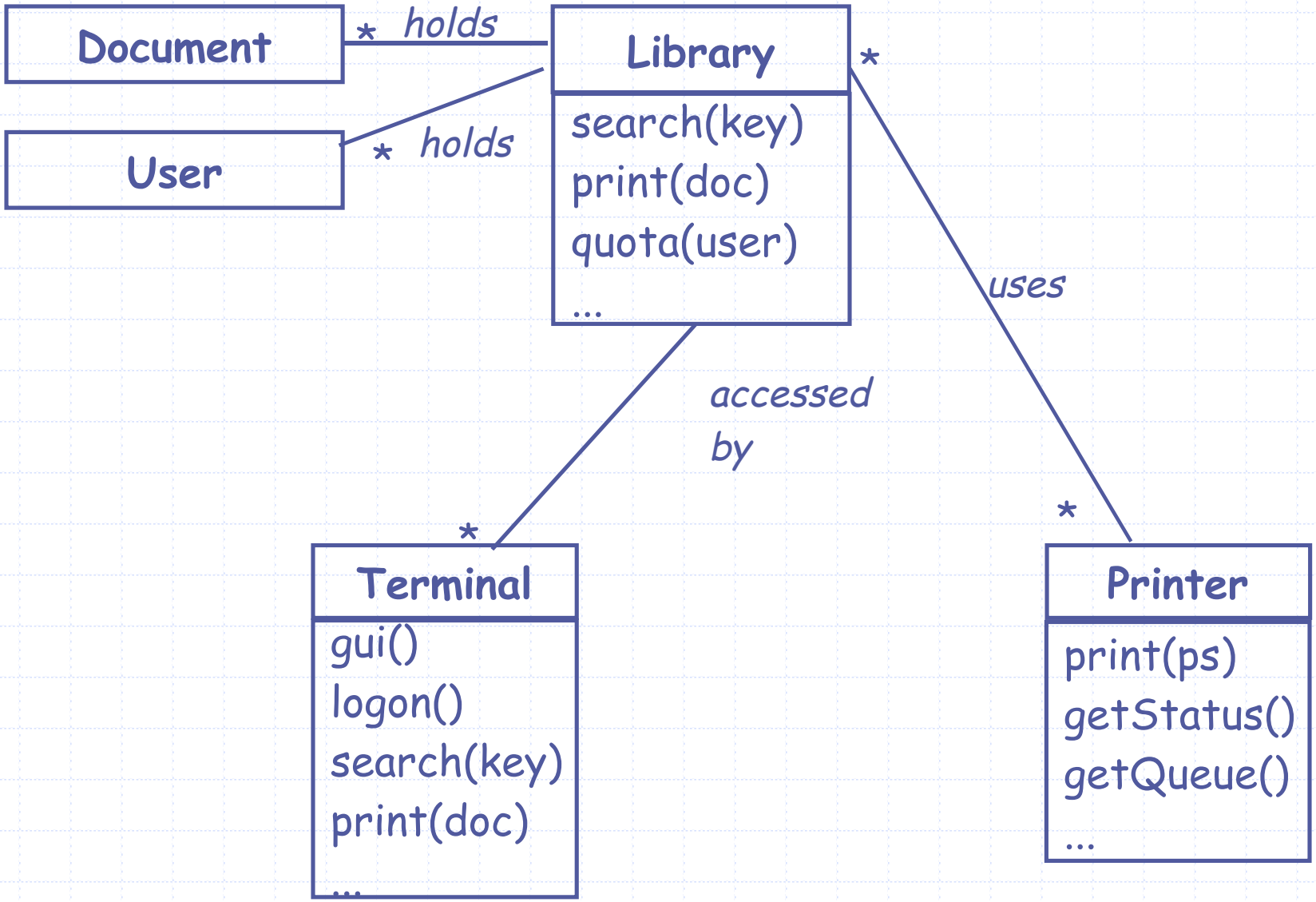
and

University of California, Irvine

Outline

- AOP and AspectJ
  - The problem and the approach
  - How people are using AOP/AspectJ
  - Facts and Numbers
- Beyond AOP
  - The need for better referencing
  - Parallels between NLs and PLs
- Warning: no clean results! (yet)

clean modular design



the code

Document

```
class Document {
  String name;
  Document(String n) { name = n; }
}
```

User

```
class User {
  String name;
  int quota;
  User(String n) { name = n; }
}
```

Terminal

```
public class Terminal {
 private Library lib;
  Thread termthread;

  Terminal (Library l) { lib = l; }

  public void logon() {}
  public void gui() {}

  public Document search(String key) {
    try {
      return lib.search(this, key);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
    return null;
  }
  public void print(Document doc) {
    try {
      lib.print(this, doc);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
  }

  void interruption() {
    termthread.interrupt();
  }

  void run() {
    termthread = Thread.currentThread();
    while (true) {

    }
  }
}
```

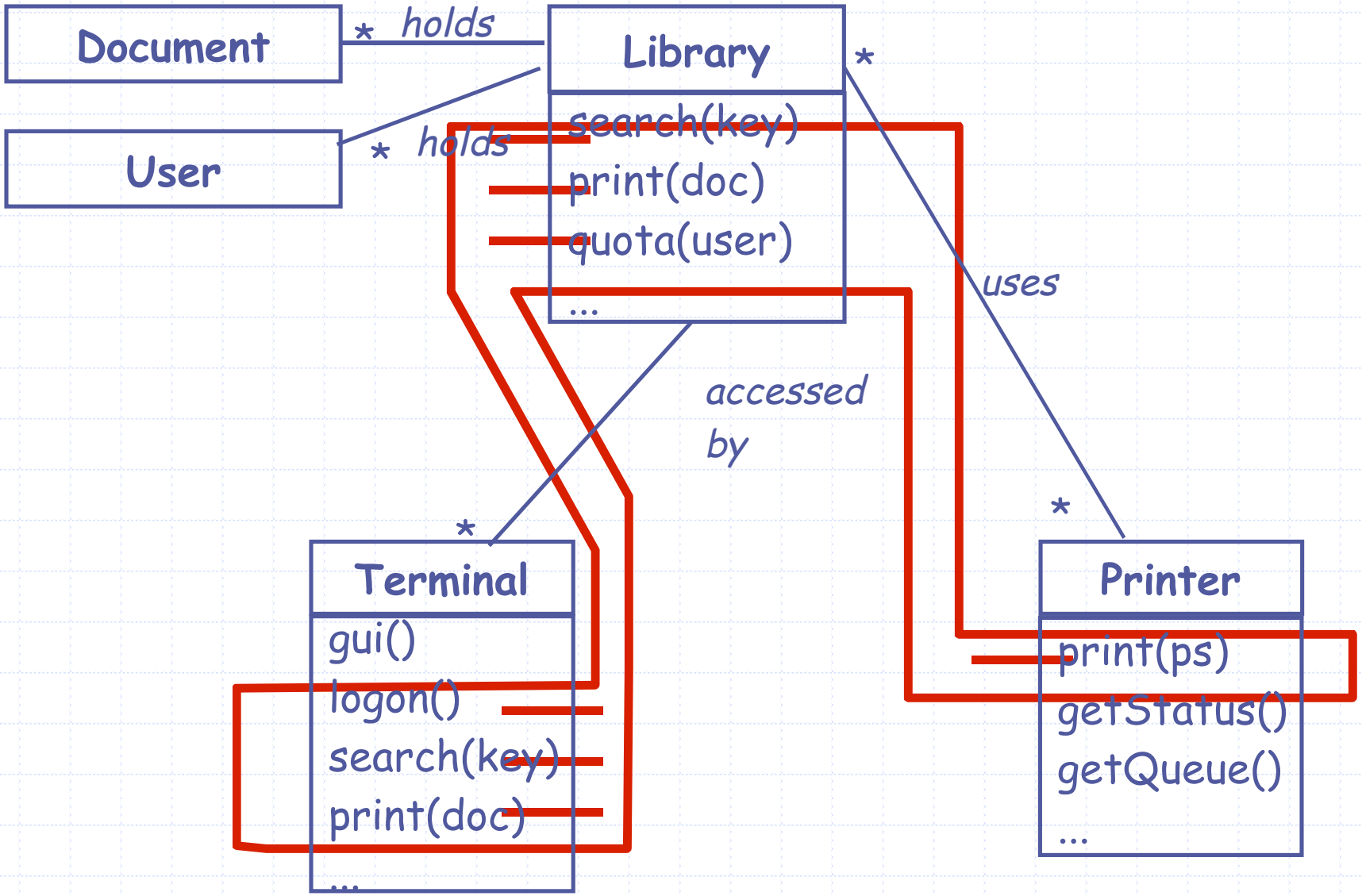
Library

```
public class Library {
  private Hashtable docs = new Hashtable();
  private Hashtable users = new Hashtable();

  private Printer printer;
  private class TerminalInfo {
    Terminal terminal;
    Thread thread;
  };
  TerminalInfo term;
  Library(Printer p) {
    printer = p;
    for (int i = 0; i < 100 ; i++) {
      Document doc = new Document("book" + String.valueOf(i));
      docs.put(doc, "book" + String.valueOf(i));
    }
    for (int i = 0; i < 100 ; i++) {
      User user = new User("user" + String.valueOf(i));
      users.put(user, "user" + String.valueOf(i));
    }
  }
  public Document search(Terminal t, String key) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return (Document)docs.get(key);
    }
    catch (Exception e) {
      System.out.println("Search interrupted at user's request");
    }
    return null;
  }
  public boolean print(Terminal t, Document doc) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return printer.print(doc);
    }
    catch (Exception e) {
      System.out.println("Print interrupted at user's request");
      printer.interruption(doc);
    }
    return false;
  }
  public String quota(Terminal t, String username) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      User user = (User)users.get(username);
      if (user != null)
        return String.valueOf(user.quota);
    }
    catch (Exception e) {
      System.out.println("Quota query interrupted at user's request");
    }
    return null;
  }

  public void interruption(Terminal t) {
    if (t == term.terminal) {
      term.thread.interrupt();
    }
  }
  public static void main(String[] args) {
    Printer printer = new Printer();
    Library lib = new Library(printer);
    Terminal term = new Terminal(lib);
  }
}
```

Printer

```
public class Printer {
  Thread pthread;
  boolean print(Document doc) {
    pthread = Thread.currentThread();
    System.out.println("Printing " + doc.name);
    try { Thread.sleep(3000); }
    catch (InterruptedException e) {
      System.out.println("Print interrupted");
      return false;
    };
    return true;
  }

  public void interruption(Document doc) {
    pthread.interrupt();
  }
}
```

user interruption



Document

```
class Document {
  String name;
  Document(String n) { name = n; }
}
```

User

```
class User {
  String name;
  int quota;
  User(String n) { name = n; }
}
```

Terminal

```
public class Terminal {
 private Library lib;
  Thread termthread;

  Terminal (Library l) { lib = l; }

  public void logon() {}
  public void gui() {}

  public Document search(String key) {
    try {
      return lib.search(this, key);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
    return null;
  }
  public void print(Document doc) {
    try {
      lib.print(this, doc);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
  }

  void interruption() {
    termthread.interrupt();
  }

  void run() {
    termthread = Thread.currentThread();
    while (true) {

    }
  }
}
```

Library

```
public class Library {
  private Hashtable docs = new Hashtable();
  private Hashtable users = new Hashtable();

  private Printer printer;
  private class TerminalInfo {
    Terminal terminal;
    Thread thread;
  };
  TerminalInfo term;
  Library(Printer p) {
    printer = p;
    for (int i = 0; i < 100 ; i++) {
      Document doc = new Document("book" + String.valueOf(i));
      docs.put(doc, "book" + String.valueOf(i));
    }
    for (int i = 0; i < 100 ; i++) {
      User user = new User("user" + String.valueOf(i));
      users.put(user, "user" + String.valueOf(i));
    }
  }
  public Document search(Terminal t, String key) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return (Document)docs.get(key);
    }
    catch (Exception e) {
      System.out.println("Search interrupted at user's request");
    }
    return null;
  }
  public boolean print(Terminal t, Document doc) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return printer.print(doc);
    }
    catch (Exception e) {
      System.out.println("Print interrupted at user's request");
      printer.interruption(doc);
    }
    return false;
  }
  public String quota(Terminal t, String username) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      User user = (User)users.get(username);
      if (user != null)
        return String.valueOf(user.quota);
    }
    catch (Exception e) {
      System.out.println("Quota query interrupted at user's request");
    }
    return null;
  }

  public void interruption(Terminal t) {
    if (t == term.terminal) {
      term.thread.interrupt();
    }
  }
  public static void main(String[] args) {
    Printer printer = new Printer();
    Library lib = new Library(printer);
    Terminal term = new Terminal(lib);
  }
}
```

locality

- "tangled code"
- code redundancy
- difficult to reason about
- difficult to change

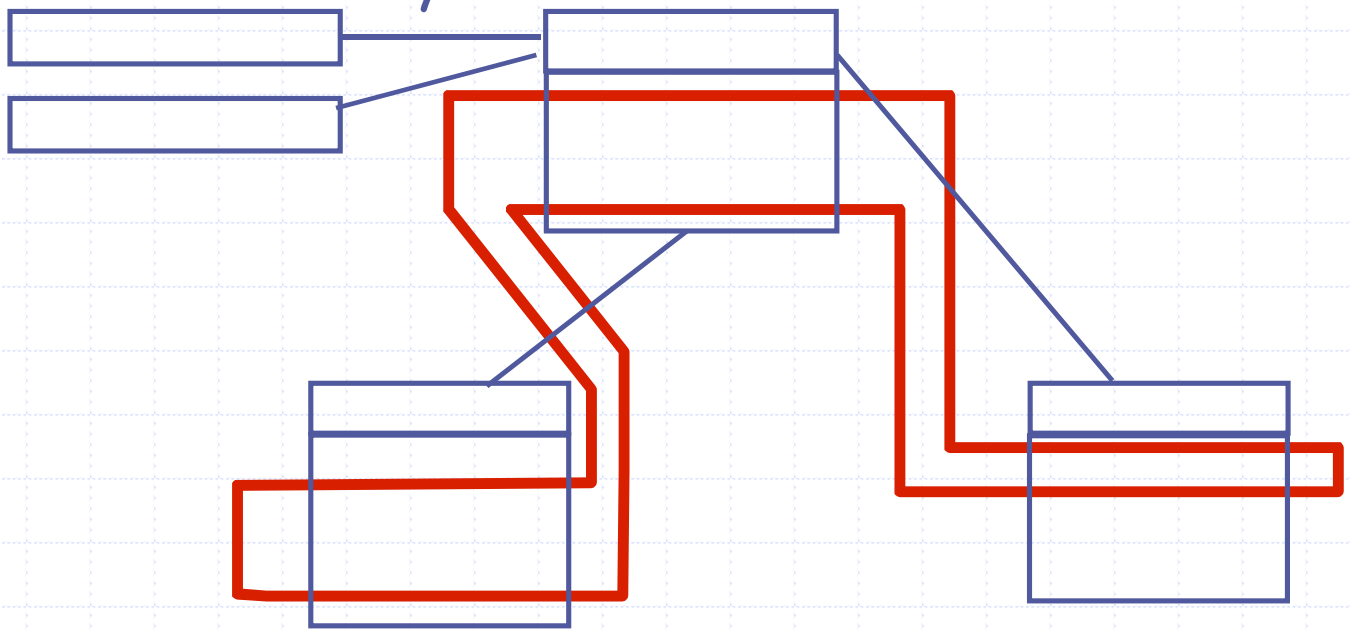
Printer

```
public class Printer {
  Thread pthread;
  boolean print(Document doc) {
    pthread = Thread.currentThread();
    System.out.println("Printing " + doc.name);
    try { Thread.sleep(3000); }
    catch (InterruptedException e) {
      System.out.println("Print interrupted");
      return false;
    };
    return true;
  }

  public void interruption(Document doc) {
    pthread.interrupt();
  }
}
```

crosscutting concerns

two concerns[1] crosscut when:

- given a "good modularity" for capturing one concern
- you can't cleanly localize the other concern



1. a concern is some part of the problem that we want to treat as a single conceptual unit

support for aspects in AspectJ

Document

```
class Document {
  String name;
  Document(String n) { name = n; }
}
```

User

```
class User {
  String name;
  int quota;
  User(String n) { name = n; }
}
```

Terminal

```
public class Terminal {
 private Library lib;
  Thread termthread;

  Terminal (Library l) { lib = l; }

  public void logon() {}
  public void gui() {}

  public Document search(String key) {
    try {
      return lib.search(this, key);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
    return null;
  }
  public void print(Document doc) {
    try {
      lib.print(this, doc);
    }
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(this);
    }
  }

  void interruption() {
    termthread.interrupt();
  }

  void run() {
    termthread = Thread.currentThread();
    while (true) {

    }
  }
}
```

Library

```
public class Library {
  private Hashtable docs = new Hashtable();
  private Hashtable users = new Hashtable();

  private Printer printer;
  private class TerminalInfo {
    Terminal terminal;
    Thread thread;
  };
  TerminalInfo term;
  Library(Printer p) {
    printer = p;
    for (int i = 0; i < 100 ; i++) {
      Document doc = new Document("book" + String.valueOf(i));
      docs.put(doc, "book" + String.valueOf(i));
    }
    for (int i = 0; i < 100 ; i++) {
      User user = new User("user" + String.valueOf(i));
      users.put(user, "user" + String.valueOf(i));
    }
  }
  public Document search(Terminal t, String key) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return (Document)docs.get(key);
    }
    catch (Exception e) {
      System.out.println("Search interrupted at user's request");
    }
    return null;
  }
  public boolean print(Terminal t, Document doc) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      return printer.print(doc);
    }
    catch (Exception e) {
      System.out.println("Print interrupted at user's request");
      printer.interruption(doc);
    }
    return false;
  }
  public String quota(Terminal t, String username) {
    term.terminal = t;
    term.thread = Thread.currentThread();
    try {
      User user = (User)users.get(username);
      if (user != null)
        return String.valueOf(user.quota);
    }
    catch (Exception e) {
      System.out.println("Quota query interrupted at user's request");
    }
    return null;
  }

  public void interruption(Terminal t) {
    if (t == term.terminal) {
      term.thread.interrupt();
    }
  }
  public static void main(String[] args) {
    Printer printer = new Printer();
    Library lib = new Library(printer);
    Terminal term = new Terminal(lib);
  }
}
```

Printer

```
public class Printer {
  Thread pthread;
  boolean print(Document doc) {
    pthread = Thread.currentThread();
    System.out.println("Printing " + doc.name);
    try { Thread.sleep(3000); }
    catch (InterruptedException e) {
      System.out.println("Print interrupted");
      return false;
    };
    return true;
  }

  public void interruption(Document doc) {
    pthread.interrupt();
  }
}
```

support for aspects in AspectJ

Document

```
class Document {
  String name;
  Document(String n) { name = n; }
}
```

User

```
class User {
  String name;
  into quota;
  User(String n) { name = n; }
}
```

Terminal

```
public class Terminal {
 private Library lib;
  Terminal (Library l) { lib = l; }

  public void logon() {}
  public void gui() {}

  public Document search(String key) {
    return lib.search(this, key);
    }
  }
  public void print(Document doc) {
      lib.print(this, doc);
  }

  void run() {
    termthread = Thread.currentThread();
    while (true) {

    }
  }
}
```

Library

```
public class Library {
  private Hashtable docs = new Hashtable();
  private Hashtable users = new Hashtable();

  private Printer printer;
  private class TerminalInfo {
    Terminal terminal;
  };
  TerminalInfo term;

  Library(Printer p) {
    printer = p;
    for (int i = 0; i < 100 ; i++) {
      Document doc = new Document("book" + String.valueOf(i));
      docs.put(doc, "book" + String.valueOf(i));
    }
    for (int i = 0; i < 100 ; i++) {
      User user = new User("user" + String.valueOf(i));
      users.put(user, "user" + String.valueOf(i));
    }
  }

  public Document search(Terminal t, String key) {
    term.terminal = t;
    return (Document)docs.get(key);
  }

  public boolean print(Terminal t, Document doc) {
    term.terminal = t;
    return printer.print(doc);
  }

  public String quota(Terminal t, String username) {
    term.terminal = t;
    User user = (User)users.get(username);
      if (user != null)
        return String.valueOf(user.quota);
  }

  public static void main(String[] args) {
    Printer printer = new Printer();
    Library lib = new Library(printer);
    Terminal term = new Terminal(lib);
  }
}
```

UserInterrupt

```
class UserRequestThreads {
  introduction (Terminal | Library | Printer) {
    Thread thread;

    void interruption() {
      thread.interrupt();
    }
  }

  static advice (Document search(String key) | void print(Document doc))
                & Terminal {
    catch (Exception e) {
      System.out.println("Operation interrupted at user's request");
      lib.interruption(thisJoinPoint.object);
    }
  }
  static advice (Terminal & (Document search(String key) |
                             void print(Document doc))) |
        (Library  & (Document search(Terminal t, String key) |
                     String quota(Terminal t, String username) |
                     boolean print(Terminal t, Document doc))) |
        (Printer  & boolean print(Document doc)) {
    before {
      thread = Thread.currentThread();
    }
  }
  static advice Library & (Document search(Terminal t, String key) |
                           String quota(Terminal t, String username)) {
    catch (Exception e) {
      System.out.println("Search interrupted at user's request");
      return null;
    }
  }
  static advice Library & boolean print(Terminal t, Document doc) {
    catch (Exception e) {
      System.out.println("Search interrupted at user's request");
      printer.interruption(doc);
      return false;
    }
  }
  static advice Printer & boolean print(Document doc) {
    catch (InterruptedException e) {
      System.out.println("Print interrupted");
      return false;
    }
  }
}
```

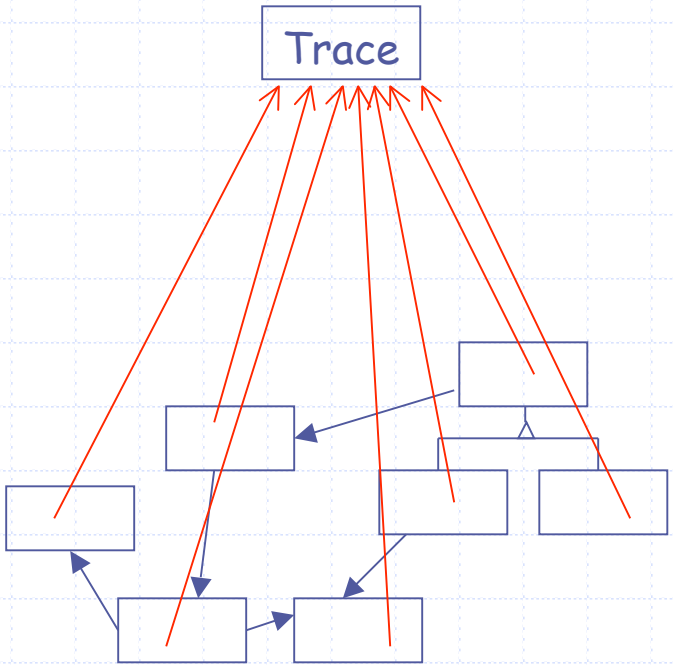
Printer

```
public class Printer {
  boolean print(Document doc) {
  System.out.println("Printing " + doc.name);
    try { Thread.sleep(3000); }
    catch (InterruptedException e) {
      System.out.println("Print interrupted");
      return false;
    };
    return true;
  }
}
```

AspectJ is…

- a small extension to Java, currently at 1.1.0 release
- an eclipse.org project
  - http://www.eclipse.org/aspectj

example: tracing

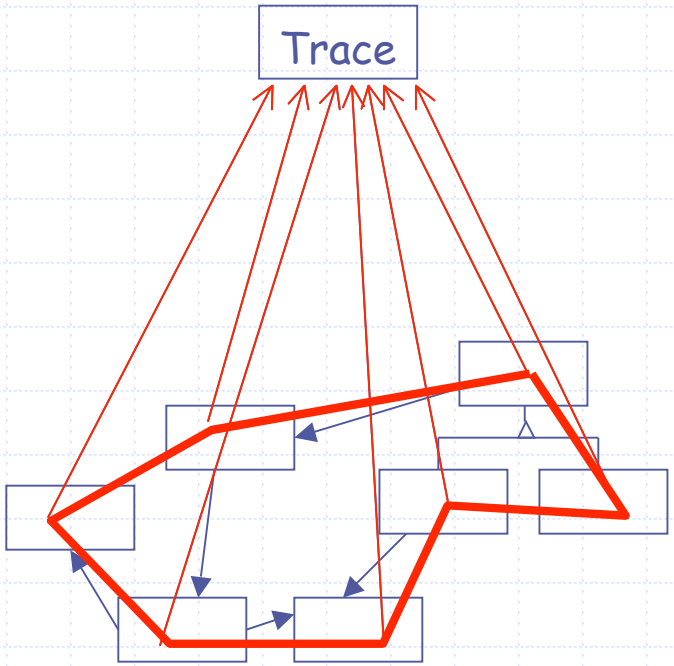



```
class Trace {
  static int TRACELEVEL = 0;
  static protected PrintStream stream = null;
  static protected int callDepth = -1;

  static void init(PrintStream _s) {stream=_s;}

  static void traceEntry (String str) {
    if (TRACELEVEL == 0) return;
    callDepth++;
    printEntering(str);
  }

  static void traceExit (String str) {
    if (TRACELEVEL == 0) return;
    callDepth--;
    printExiting(str);
  }
  ...
}
```

```
class Point {
  void set(int x, int y) {
    Trace.traceEntry("Point.set");
    _x = x; _y = y;
    Trace.traceExit("Point.set");
  }
}
```

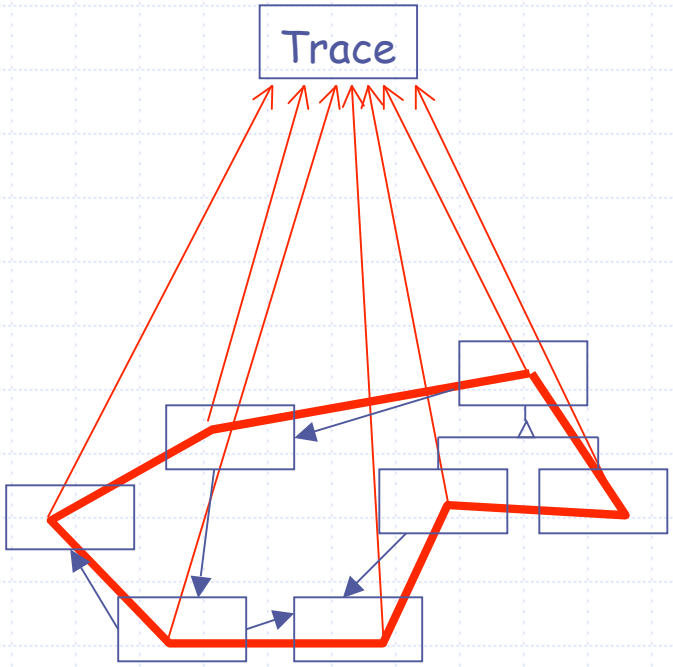
what is the crosscut?

the modules use the trace facility in a consistent way:
when entering the methods and
when exiting the methods



this line is about interacting with the trace facility

tracing as an aspect (in aspectj)




```
aspect TraceMyClasses {

  pointcut traceCut(): mypackage.* &&
                       public * *(..);

  before(): traceCut() {
    Trace.traceEntry(tjp.className +"."+
                     tjp.methodName);
  }
  finally(): traceCut() {
    Trace.traceExit(tjp.className +"."+
                    tjp.methodName);
  }
}
```

plug

- plug in:

```
ajc Point.java Line.java
    Trace.java TraceMyClasses.java
```

- unplug:

```
ajc Point.java Line.java
```

  - turn debugging on/off without editing classes
  - debugging disabled with no runtime cost
  - can save debugging code between uses

How are people using it?

- **debugging and instrumentation Aspects** such as tracing, logging, testing, profiling, monitoring and asserting.
- **program construction Aspects** such as mixins, multiple-inheritance and views;
- **configuration Aspects** such as managing the specifics of using different platforms and choosing appropriate name spaces for property management;
- **enforcement and verification Aspects** such as making sure the types of a framework are used appropriately, components' contract validation and ensuring best programming practices;
- **operating Aspects** such as synchronization, caching, persistence, transaction management, security and load balancing;
- **failure handling Aspects** such as redirecting a failed call to a different service;

- "pluggability" being perceived as major advantage

History, Facts and Numbers

- started as small research project ~1995
  - CLOS, Demeter, my thesis, and others
- funded mainly by DARPA for 5 years
- today: ~1000 people in users list
- AOP paper (ECOOP'97): #4 (1997), #46 (ever)
  - http://citeseer.nj.nec.com/source-1997.html
  - http://citeseer.nj.nec.com/source.html
- ACM-sponsored conference: AOSD
  - http://aosd.net/conference.html
- JavaWorld's Editors' Choice Award: "Most innovative Java Product or Technology"
  - http://www.javaworld.com/javaworld/jw-06-2003/jw-0609-eca_p.html
- several AOP/AspectJ books in amazon.com
- …

Going Forward: Example from a modem (1)

```
/**
 * encodeStream converts a given stream of bytes into sounds.
 * @param input the stream of bytes to encode
 * @param output the stream of audio samples representing the input
 */
static void encodeStream(InputStream input, OutputStream output){
      int readindex = 0;
      byte[] buff = new byte[kBytesPerDuration];
      while( (readindex = input.read(buff)) == kBytesPerDuration){
          output.write(Encoder.encodeDuration(buff));
      }
      if (readindex > 0){
          for (int i=readindex; i < kBytesPerDuration; i++){
              buff[i] = 0;
          }
          output.write(Encoder.encodeDuration(buff));
      }
}
```

Example from a modem (2)

```
/**
 * encodeStream converts a given stream of bytes into sounds.
 * @param input the stream of bytes to encode
 * @param output the stream of audio samples representing the input
 */
static void encodeStream(InputStream input, OutputStream output){
      . Create an integer called readindex and initialize it to zero.
      . Create an array of kBytesPerDuration bytes called buff.
      . A loop begins:
        . Request the service read from input, with argument buff;
          set readindex to the return value of this service.
      . If readindex is equal to kBytesPerDuration, then
        . Request the service write from output;
          the argument to this service is the return value of
            . Request the service encodeDuration from Encoder,
              with argument buff.
      End of loop.
   . If readindex is greater than 0 then
        . Set to zero all positions of buff starting at readindex.
        . Request the service write from output;
          the argument to this service is the return value of
           . Request the service encodeDuration from Encoder,
             with argument buff.
```

terrible!

Example from a modem (3)

```
/**
  * encodeStream converts a given stream of bytes into sounds.
  * @param input the stream of bytes to encode
  * @param output the stream of audio samples representing the input
  */
 static void encodeStream(InputStream input, OutputStream output){
       while there is data in the input stream:
         read the first kBytesPerDuration bytes from it.
         perform encodeDuration on those bytes.
         write the result of this last operation into the output stream.
       if, after reading the input stream, the number
          of bytes read is less than kBytesPerDuration, then patch
          the resulting byte array with zeros before continuing.
```

anaphora

Toward a Naturalistic Language

- Vision:
  - Executable *naturalistic* specifications, i.e. programming system closer to people's natural language expressions and program organizations
  - One small general-purpose language upon which several domain-specific idioms will be built
    - Language (wordless mechanisms) vs. Vocabulary (words)
- The road ahead:
  - Better structural and reflective references
  - Temporal references
  - Quantification
- Immediate plan:
  - Borrow heavily from Linguistics: models of NLs
  - Study existing programs with "naturalistic glasses"
  - Brainstorm towards a prototype

The core of AspectJ
(in the light of Naturalistic Programming)

- simple temporal references
  - before certain points in execution
  - after certain points in execution
  - join point model provides the temporal 'hooks'
- somewhat simple declarative reflection model
  - refer to and manipulate classes, methods, etc.
- this allows novel program text organizations
  - e.g. tracing is like a separate chapter in imaginary book that's the application

AOP credits

- Gregor Kiczales
- John Lamping
- Karl Lieberherr
- Anurag Mendhekar
- Jean-Marc Loingtier
- Chris Maeda
- John Irwin
- Venkatesh Choppella
- Jim Hugunin
- Erik Hilsdale
- Mik Kersten
- Martin Lippert
- Bill Griswold
- Mark Skipper
- Jeff Palm
- and many more!