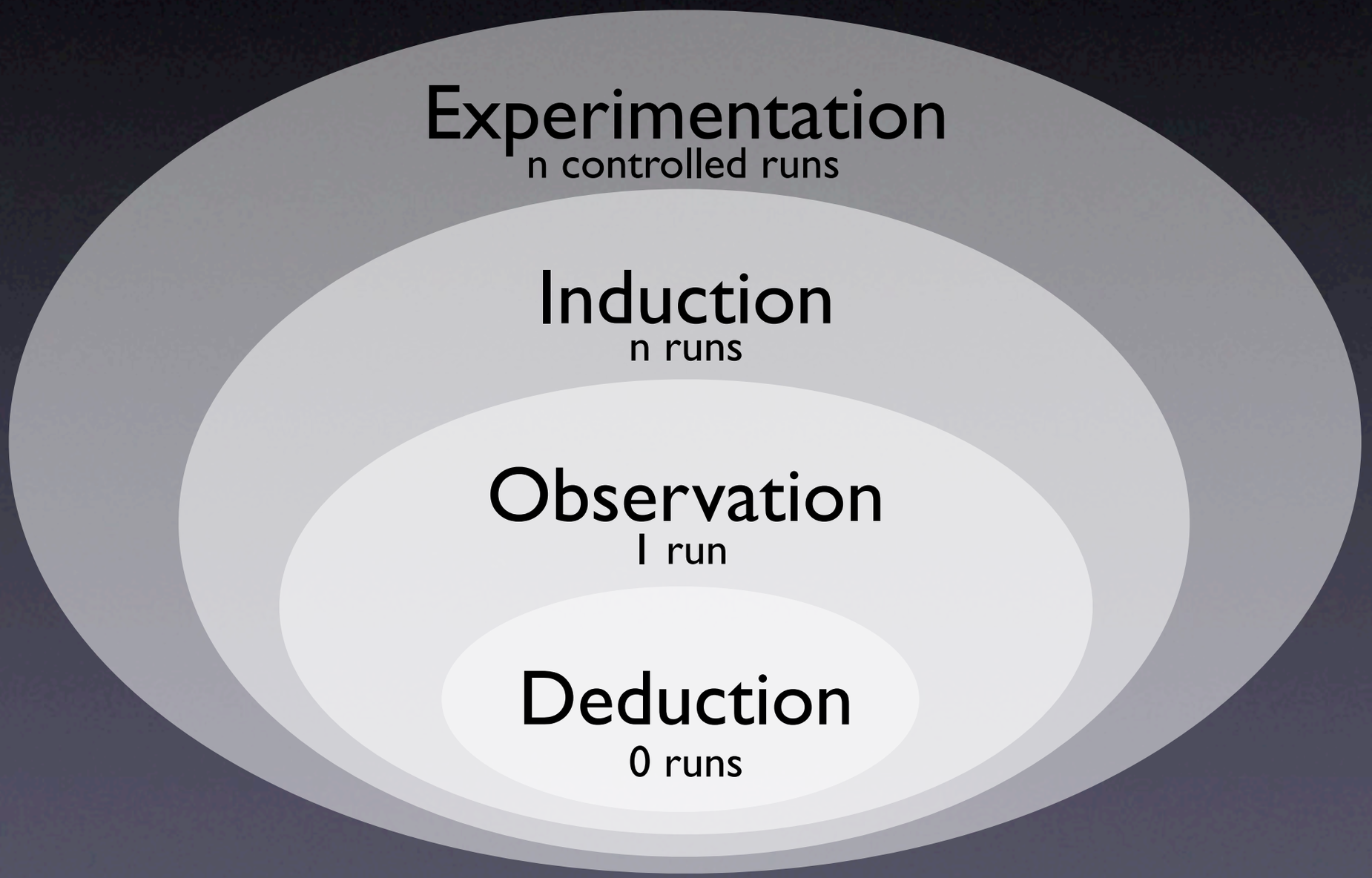
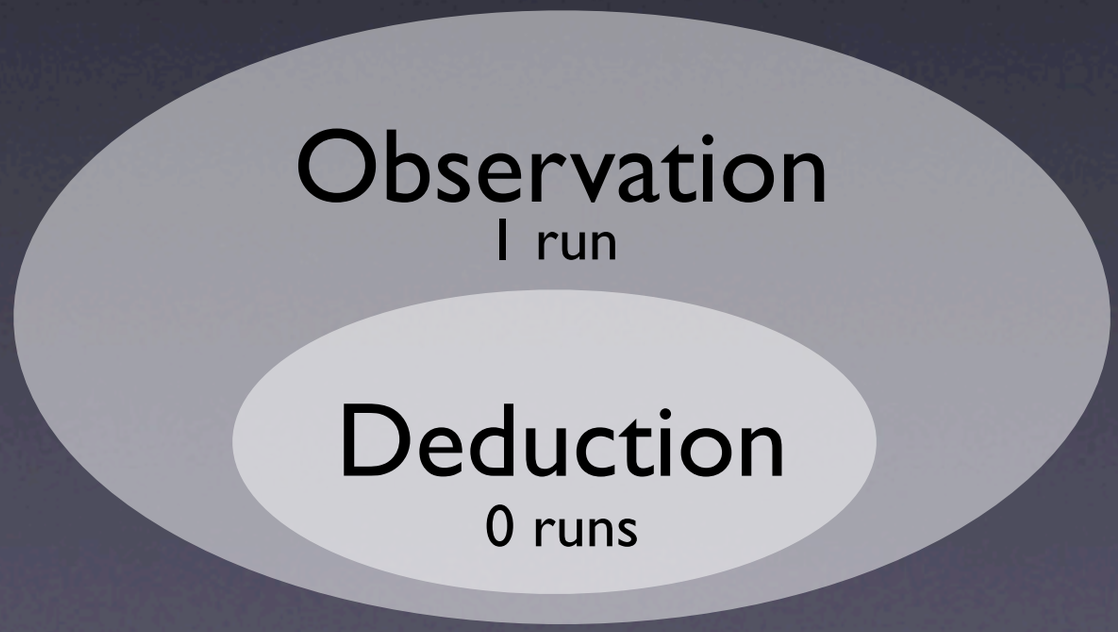
Observing Facts

Andreas Zeller

Reasoning about Runs



Reasoning about Runs



Principles of Observation

- Don't interfere.
- Know what and when to observe.
- Proceed systematically.

Logging execution

- General idea: Insert *output statements* at specific places in the program
- Also known as *printf debugging*

Printf Problems

- Clobbered code
- Clobbered output
- Slow down
- Possible loss of data (due to buffering)

Better Logging

- Use standard formats
- Make logging optional
- Allow for variable granularity
- Be persistent

Logging Functions

- Have specific functions for logging (e.g. dprintf() to print to a specific logging channel)
- Have specific *macros* that can be turned on or off–for focusing as well as for production code

Logging Frameworks

- Past: home-grown logging facilities
- Future: *standard libraries* for logging
- Example: The LOGFORJ framework

LOGFORJ

```
// Initialize a logger.
final ULogger logger =
    LoggerFactory.getLogger(TestLogging.class);

// Try a few logging methods
public static void main(String args[]) {
    logger.debug("Start of main()");
    logger.info ("A log message with level set to INFO");
    logger.warn ("A log message with level set to WARN");
    logger.error("A log message with level set to ERROR");
    logger.fatal("A log message with level set to FATAL");

    new TestLogging().init();
}
```

Customizing Logs

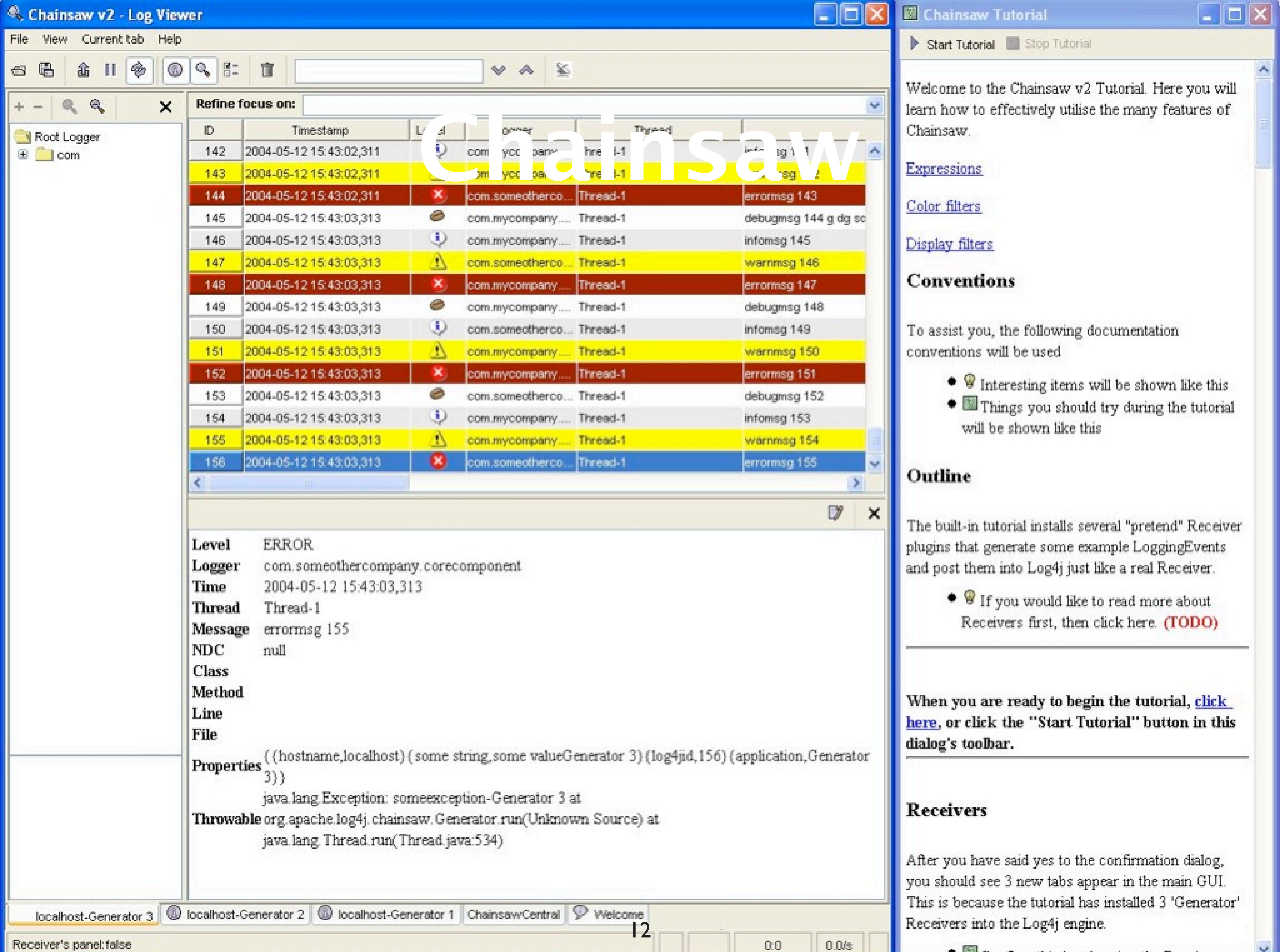
```
# Set root logger level to DEBUG and its only appender to A1.
log4j.rootLogger=DEBUG, A1

# A1 is set to be a ConsoleAppender.
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=\
%d [%t] %-5p %c %x - %m%n
```

```
2005-02-06 20:47:31,508 [main] DEBUG  TestLogging - Start of
main()
2005-02-06 20:47:31,529 [main] INFO   TestLogging - A log
message with level set to INFO
```

Chainsaw



Chainsaw v2 - Log Viewer
File View Current tab Help
Refine focus on:
Root Logger
com
ID Timestamp Level Logger Thread Message
144 2004-05-12 15:43:02,311 com.someotherco... Thread-1 errormsg 143
145 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 debugmsg 144 g dg sc
146 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 infomsg 145
147 2004-05-12 15:43:03,313 com.someotherco... Thread-1 warnmsg 146
148 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 errormsg 147
149 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 debugmsg 148
150 2004-05-12 15:43:03,313 com.someotherco... Thread-1 infomsg 149
151 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 warnmsg 150
152 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 errormsg 151
153 2004-05-12 15:43:03,313 com.someotherco... Thread-1 debugmsg 152
154 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 infomsg 153
155 2004-05-12 15:43:03,313 com.mycompany.... Thread-1 warnmsg 154
156 2004-05-12 15:43:03,313 com.someotherco... Thread-1 errormsg 155
Level ERROR
Logger com.someothercompany.corecomponent
Time 2004-05-12 15:43:03,313
Thread Thread-1
Message errormsg 155
NDC null
Class
Method
Line
File
Properties {{hostname,localhost}{some string,some valueGenerator 3}{log4jid,156}{application,Generator 3}}
Throwable java.lang.Exception: someexception-Generator 3 at org.apache.log4j.chainsaw.Generator.run(Unknown Source) at java.lang.Thread.run(Thread.java:534)
localhost-Generator 3 localhost-Generator 2 localhost-Generator 1 ChainsawCentral Welcome
Receiver's panel:false
0:0 0.0/s
Chainsaw Tutorial
Start Tutorial Stop Tutorial
Welcome to the Chainsaw v2 Tutorial. Here you will learn how to effectively utilise the many features of Chainsaw.
Expressions
Color filters
Display filters
Conventions
To assist you, the following documentation conventions will be used
Interesting items will be shown like this
Things you should try during the tutorial will be shown like this
Outline
The built-in tutorial installs several "pretend" Receiver plugins that generate some example LoggingEvents and post them into Log4j just like a real Receiver.
If you would like to read more about Receivers first, then click here. (TODO)
When you are ready to begin the tutorial, click here, or click the "Start Tutorial" button in this dialog's toolbar.
Receivers
After you have said yes to the confirmation dialog, you should see 3 new tabs appear in the main GUI. This is because the tutorial has installed 3 'Generator' Receivers into the Log4j engine.

Logging with Aspects

- Basic idea: Separate concerns into individual syntactic entities *(aspects)*
- Aspect code *(advice)* is *woven* into the program code at specific places *(join points)*
- The same aspect code can be woven into multiple places *(pointcuts)*

A Logging Aspect

```
public aspect LogBuy {
    pointcut buyMethod():
        call(public void Article.buy());
    before(): buyMethod() {
      System.out.println("Entering Article.buy()")
    }
    after(): buyMethod() {
      System.out.println("Leaving Article.buy()")
    }
}
```

```
$ ajc logBuy.aj Article.java
$ java Article
```

Using Pointcuts

```
public aspect LogArticle {
  pointcut allMethods():
    call(public * Article.*(..));
  before(): allMethods() {
    System.out.println("Entering " + thisJoinPoint)
  }
  after(): allMethods() {
    System.out.println("Leaving " + thisJoinPoint)
  }
}
```

Aspect Arguments

```
public aspect LogMoves {
    pointcut setP(Line a_line, Point p):
        call(void a_line.setP*(p));

    after(Line a_line, Point p): setP(a_line, p) {
        System.out.println(a_line +
                           " moved to " + p + ".");
    }
}
```

Logging at the binary level

- The PIN framework provides *dynamic instrumentation* of x86 executables

```
int main(int argc, char * argv[])
{
    trace = fopen("itrace.out", "w");

    // Initialize pin
    PIN_Init(argc, argv);

    // Register Instruction to be called to instrument insns
    INS_AddInstrumentFunction(Instruction, 0);

    // Register Fini to be called when the application exits
    PIN_AddFiniFunction(Fini, 0);

    // Start the program, never returns
    PIN_StartProgram();

    return 0;
}
```

```
// This function is called before every instruction is executed
// and prints the IP
VOID printip(VOID *ip) { fprintf(trace, "%p\n", ip); }

// Pin calls this function every time
// a new instruction is encountered
VOID Instruction(INS ins, VOID *v)
{
    // Insert a call to printip before every instruction,
    // and pass it the IP
    INS_InsertCall(ins, IPOINT_BEFORE, (AFUNPTR)printip,
        IARG_INST_PTR, IARG_END);
}
```

```
$ cd pin-2.0/ManualExamples
$ make itrace
$ ../Bin/pin -t itrace -- /bin/ls
atrace.C     inscount0.C  _insprofiler.C  itrace.o
staticcount.C…
$ head itrace.out    # output first 10 lines
0x40000c20
0x40000c22
0x40000c70
0x40000c71
0x40000c73
0x40000c74
0x40000c75
0x40000c76
0x40000c79
0x40011d9b
$ wc -l itrace.out
501585
```

Observation Tools

- Getting started fast – without altering the program code at hand
- Flexible observation of arbitrary events
- Transient sessions – no code is written

Debuggers

- Execute the program and make it stop under specific conditions
- Observe the state of the stopped program
- Change the state of the program

A Debugging Session

```
static void shell_sort(int a[], int size)
{
    int i, j;
    int h = 1;
    do {
        h = h * 3 + 1;
    } while (h <= size);
    do {
        h /= 3;
        for (i = h; i < size; i++)
        {
            int v = a[i];
            for (j = i; j >= h && a[j - h] > v; j -= h)
                a[j] = a[j - h];
            if (i != j)
                a[j] = v;
        }
    } while (h != 1);
}
```

More Features

- Control environment
- Post mortem debugging
- Logging data
- Fix and continue

Debugger Caveats

- A debugger is a tool, not a toy!

More on Breakpoints

- Data breakpoints (watchpoints)
- Conditional breakpoints

Querying in COCA

Events	Data
type	name
port	type
func	val
chrono	addr
depth	size
line	linedecl
file	filedecl

Example Queries

```
[coca] current_var(Name, val=42).
Name = x0
Name = x1
[coca] fget(func=shell_sort and line=Ln), \
       current_var(Name, val=0).
Name = a[2]  Ln = <int i, j;>
Name = v     Ln = <int v = a[i]>
Name = a[0]  Ln = <a[j] = v>
[coca] fget(line=Ln), current_var(a, val=array(-,-,0,...)).
Ln = <a = malloc(...)>
[coca]
```

fget() sets breakpoints, current_var() queries data

Visualizing Data

DDD: /public/source/programming/ddd-3.2/ddd/cxxtest.C

File Edit View Program Commands Status Source Data Help

0: list- self

Set Undisp

1: list
(List *) 0x804df80

*()

value = 85
self = 0x804df80
next = 0x804df90

self.

next

value = 86
self = 0x804df90
next = 0x804df...

self.

next

next

```
    list->next                  = new List(a_global + start++);
    list->next->next            = new List(a_global + start++);
    list->next->next->next      = list;

    (void) list;                // Display this
    delete list;
    delete list->next;
    delete list;
}

// Test
void lis
{
    list
}

//------
void ref
{
    date
    dele
    date
}
```

(List *) 0x804df80

DD
Run
Interrupt
Step Stepi
Next Nexti
Until Finish
Kill
own
edo
lake

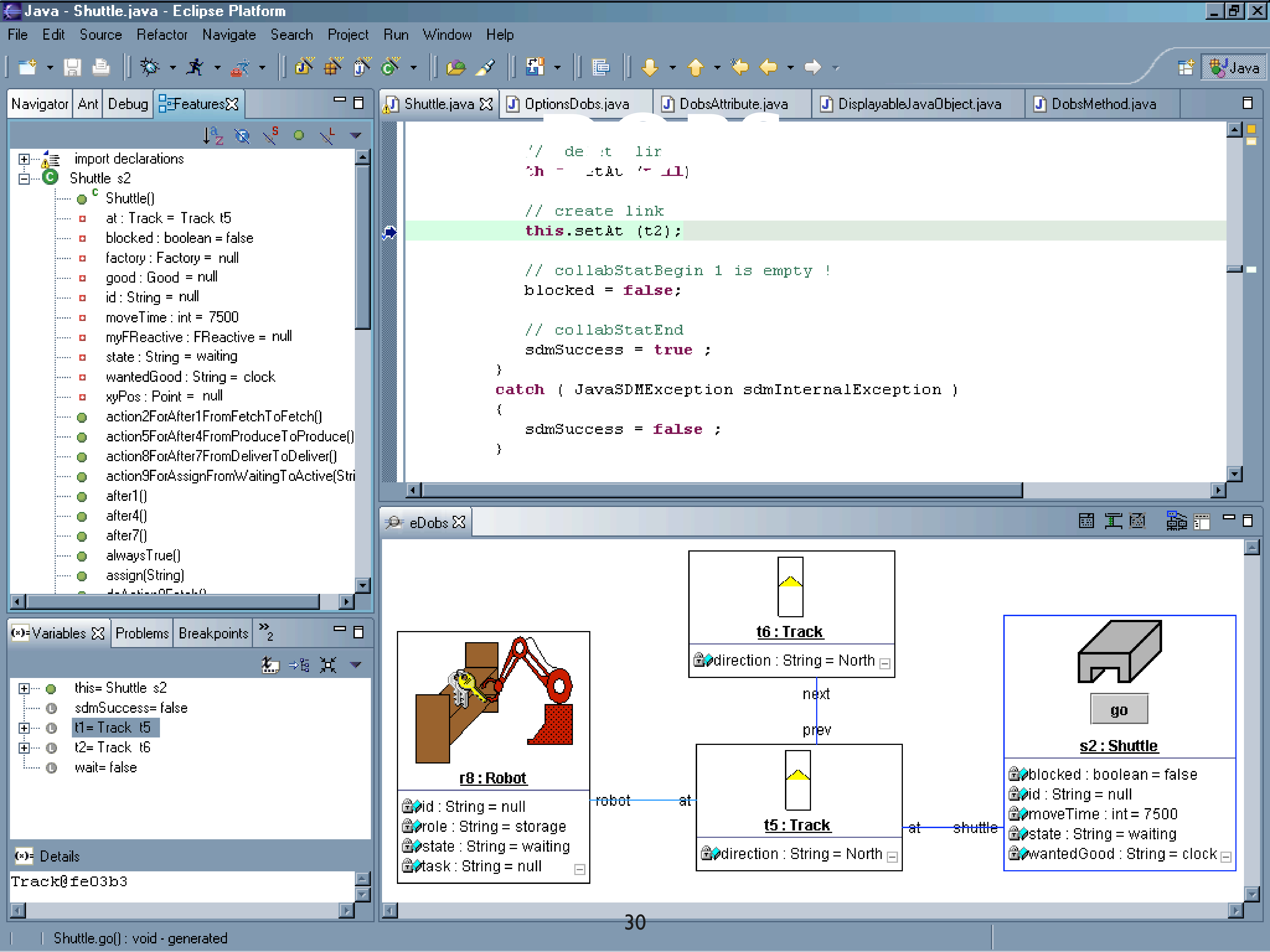
DDD Tip of the Day #5

If you made a mistake, try **Edit→Undo**. This will undo the most recent debugger command and redisplay the previous program state.

Close Prev Tip Next Tip

```
(gdb) graph display *(list->next->next->self) dependent on 4
(gdb)
```

list = (List *) 0x804df80


Java - Shuttle.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Navigator Ant Debug Features
import declarations
Shuttle s2
Shuttle()
at : Track = Track t5
blocked : boolean = false
factory : Factory = null
good : Good = null
id : String = null
moveTime : int = 7500
myFReactive : FReactive = null
state : String = waiting
wantedGood : String = clock
xyPos : Point = null
action2ForAfter1FromFetchToFetch()
action5ForAfter4FromProduceToProduce()
action8ForAfter7FromDeliverToDeliver()
action9ForAssignFromWaitingToActive(Stri
after1()
after4()
after7()
alwaysTrue()
assign(String)
Shuttle.java OptionsDobs.java DobsAttribute.java DisplayableJavaObject.java DobsMethod.java
// create link
this.setAt (t2);
// collabStatBegin 1 is empty !
blocked = false;
// collabStatEnd
sdmSuccess = true ;
}
catch (JavaSDMException sdmInternalException)
{
sdmSuccess = false ;
}
Variables Problems Breakpoints
this= Shuttle s2
sdmSuccess= false
t1= Track t5
t2= Track t6
wait= false
Details
Track@fe03b3
eDobs
t6 : Track
direction : String = North
next
prev
r8 : Robot
id : String = null
role : String = storage
state : String = waiting
task : String = null
robot
at
t5 : Track
direction : String = North
at
shuttle
go
s2 : Shuttle
blocked : boolean = false
id : String = null
moveTime : int = 7500
state : String = waiting
wantedGood : String = clock
Shuttle.go() : void - generated

Concepts (2)

- Logging functions can be turned on or off (and may even remain in the source code)
- Aspects elegantly keep all logging code in one place
- Debuggers allow flexible + quick observation of arbitrary events

Concepts

- ★ Logging functions (”printf debugging”) are easy to use, but clobber code and output
- ★ To encapsulate and reuse debugging code, use dedicated logging functions or aspects

Concepts (3)

- To observe the final state of a crashing program, use a debugger
- Advanced debuggers allow to query events in a declarative fashion…
- …as well as visualizing events and data