TRACEABLE ALGORITHMS

Prasun Dewan
Department of Computer Science
University of North Carolina at Chapel Hill
dewan@cs.unc.edu
Code available at: https://github.com/pdewan/ColabTeaching
PRE-REQUISITES

- Model-Interactor Separation
Algorithm vs. Program

- Description of solution to a problem.
- Can be in any “language”
  - graphical
  - natural or programming language
  - natural + programming language (pseudo code)
- Can describe solution to various levels of detail
  - A program is an algorithm
  - An algorithm may not be a program
- Level of detail depends on the task of the reader
  - If debugging or maintaining, then depends on which aspect is faulty or being changed
  - If describing solution depends on what is considered algorithm challenge
Please enter an input line or quit or history
The woods are lovely dark and deep
The woods are lovely dark and deep
Please enter an input line or quit or history
But I have promises to keep
But I have promises to keep
Please enter an input line or quit or history
And miles to go before I sleep
And miles to go before I sleep
Please enter an input line or quit or history
history
The woods are lovely dark and deep, But I have promises to keep, And miles to go before I sleep

Please enter an input line or quit or history
quit
Quitting application

Algorithm/program separation more useful in monolithic or modular program?
public class MonolithicEchoer {  
protected static List<String> history = new ArrayList();
public static void main(String[] anArgs) {  
    for (;;) {  
        System.out.println(PROMPT);
        Scanner scanner = new Scanner(System.in);
        String nextInput = scanner.nextLine();
        if (nextInput.equals(QUIT)) {
            processQuit();
            break;
        } else if (nextInput.equals(HISTORY))
            printHistory();
        else
            processInput(nextInput);
    }
}
protected static void processInput(String anInput) {
    String aFeedback = EchoUtilities.echo(anInput);
    addToHistory(aFeedback);
    displayOutput(aFeedback);
}
protected static void displayOutput(String newValue) {
    System.out.println(newValue);
}
protected static void addToHistory(String newValue) {
    history.add(history.size(), newValue);
}
**How Useful in More Modular Program?**

```java
public class ASimpleList<ElementType> implements SimpleList<ElementType> {
    List<ElementType> simpleList = new ArrayList();
    List<ListObserver<ElementType>> observers = new ArrayList();

    public void add(ElementType anElement) {
        simpleList.add(simpleList.size(), anElement);
    }

    public void observableAdd(int anIndex, ElementType anElement) {
        add(anIndex, anElement);
        notifyAdd(anIndex, anElement);
    }

    public void notifyAdd(List<ListObserver<ElementType>> observers, int index, ElementType newValue) {
        for (ListObserver<ElementType> observer : observers) {
            observer.elementAdded(index, newValue);
        }
    }
}
```
How Useful in More Modular Program?

```java
public class AnEchoInteractor implements EchoerInteractor {
    protected SimpleList<String> history;
    public AnEchoInteractor(SimpleList<String> aHistory) {
        history = aHistory;
    }
    ...
    protected void processInput(String anInput) {
        addToHistory(computeFeedback(anInput));
    }
    protected void addToHistory(String newValue) {
        history.observableAdd(newValue);
    }
    public void elementAdded(int anIndex, Object aNewValue) {
        displayOutput(history.get(anIndex));
    }
    protected void displayOutput(String newValue) {
        System.out.println(newValue);
    }
    ...
```
How Useful in More Modular Program?

```java
public class AnEchoComposerAndLauncher implements EchoerComposerAndLauncher {
    protected SimpleList<String> history;
    protected EchoerInteractor interactor;
    // factory method
    protected SimpleList<String> createHistory() {
        return new ASimpleList();
    }
    // factory method
    protected EchoerInteractor createInteractor() {
        return new AnEchoInteractor(history);
    }
    protected void connectModelInteractor() {
        interactor = createInteractor();
        history.addObserver(interactor);
    }
    ...
```

Modularity scatters algorithm among multiple objects

Need for higher-level algorithm more in multi-class programs

With observer pattern and interfaces sometimes algorithm not known until runtime
Which Comes First?

Algorithm → Program

Top-down, bottom-up, middle-out
SEPARATE?

Algorithm

Program

Can get inconsistent

Embellish the program with the algorithm
IN-LINE ALGORITHM

```java
protected static void processInput(String anInput) {
    // received input
    String aFeedback = EchoUtilities.echo(anInput);
    addToHistory(aFeedback);
    // added input to history
    displayOutput(aFeedback);
    // displayed the input
}
```

Can extract comments from code to view algorithm

Do not get a linear path from scattered objects
protected static void processInput(String anInput) {
    System.out.println("received input");
    EchoUtilities.echo(anInput);
    addToHistory(aFeedback);
    System.out.println("added input to history");
    displayOutput(aFeedback);
    System.out.println("displayed the input");
}

Can get a linear path

Cannot disable them easily

Cannot separate them from other output
TRACING WITH DEBUGGER

- Debugger makes it difficult to test race conditions
- Cannot see the history of actions
- Break points do not transfer to another computer
- No static documentation
Logging Frameworks

- Log rather than print traces
- Can display selected portions of the log
- Can separate log output from the rest

Will describe log framework developed before Java’s
Special Class for Each Algorithm Step/Event

- Trace objects and source code in separate packages
- Algorithm steps can be in separate packages
- Algorithm steps associated with checkers based on event and source
- Concrete events vs. Abstract Classes
- Each trace event object has a source or announcer

- Info vs. Warnings vs Error
- Settings for checkers

- Concrete events vs. Abstract Classes
- Each trace event object has a source or announcer

- Algorithm steps can be in separate packages
- Algorithm steps associated with checkers based on event and source
SOURCE CLASS FILTERING

Tracer.showInfo(true);
Tracer.setImplicitPrintKeywordKind(ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setMessagePrefixKind(MessagePrefixKind.FULL_CLASS_NAME);
TraceableInfo.setPrinttraceable(true);
TraceableInfo.setPrintSource(true);
TraceableInfo.setPrintTime(true);
TraceableInfo.setPrintThread(true);
Tracer.setKeywordPrintStatus(ASimpleList.class, true)

All events fired by (instances of) ASimpleList.class

Can enumerate multiple classes

Alternative to class-based filtering?

Why ListEditMade and ListEditNotified and not other events

All events fired by (instances of) ASimpleList.class
SEPARATE? (REVIEW)

Algorithm

Program

Can get inconsistent

Embellish the program with the algorithm
Special Class for Each Algorithm Step/Event

Each trace event object has a source or announcer
**Source Class Filtering**

Tracer.showInfo(true);
Tracer.setImplicitPrintKeywordKind(ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setMessagePrefixKind(MessagePrefixKind.FULL_CLASS_NAME);
TraceableInfo.setPrintTraceable(true);
TraceableInfo.setPrintSource(true);
TraceableInfo.setPrintTime(true);
TraceableInfo.setPrintThread(true);
Tracer.setKeywordPrintStatus(ASimpleList.class, true)

Why ListEditMade and ListEditNotified and not other events

All events fired by (instances of) ASimpleList.class

Can enumerate multiple classes

Alternative to source-based filtering?
**Event CLASS Filtering**

TracingLaunchEventClass [Java Application] D:\Program Files\Java\jdk1.7.0_51\bin\javaw.exe (Sep 3, 2014, 9:30:29 AM)

I***Tracer: showInfo = true
Please enter an input line or quit or history

Woods

Woods
Please enter an input line or quit or history

Tracer.showInfo(true);
Tracer.setImplicitPrintKeywordKind (ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setMessagePrefixKind (MessagePrefixKind.FULL_CLASS_NAME);
TraceableInfo.setPrintSource(true);
TraceableInfo.setPrintTime(true);
TraceableInfo.setPrintThread(true);
Tracer.setKeywordPrintStatus(ListEditMade.class, true);
Tracer.setKeywordPrintStatus(ListEditObserved.class, true);

All events of type ListEditMade or ListEditObserved

Can be announced by different sources

Alternative (source/event) class-based filtering?
PACKAGE-BASED FILTERING

All events of types that are in the package of ListEditMade;

Filtering by class and package in other contexts?

trace.echo

EchoTraceChecker.java
ListEditDisplayed.java
ListEditInfo.java
ListEditInput.java
ListEditMade.java

trace.echo.modular
EchoTracerSetter.java
ListEditNotified.java
ListEditObserved.java
ModularEchoTraceCheck

OperationName.java

Tracer.showInfo(true);
Tracer.setImplicitPrintKeywordKind(ImplicitKeywordKind.OBJECT_PACKAGE_NAME);

TraceableInfo.setPrintSource(true);
TraceableInfo.setPrintTime(true);
TraceableInfo.setPrintThread(true);

Tracer.setKeywordPrintStatus(ListEditMade.class, true);

Woods


Woods


Woods


Please enter an input line or quit or history

Woods

trace.echo

trace.echo.modular
**ASSERTIONS**

```java
public double getBmi() {
    assert weight > 0 && height > 0: "height and weight should be >0";
    return weight/(height*height);
}
```

Assertion error is like exception, but it can be disabled

Can enable/disable assertions for specific classes and packages

*Enable assertions for MainClass*

*Disable assertions for bus.uigen package*

Similarity between trace objects and assertions is not a coincidence as both support disablable testing

State vs events
Trace Object vs. Events

a la event type, class of object
a la event firing, source object
computed automatically
a la event parameters

Announcing a trace object is “asserting” an algorithm event
**Example Traceable Event Class**

```java
public class ListEditInput extends ListEditInfo{
    public ListEditInput(String aMessage,
                           OperationName anOperationName, int anIndex,
                           Object anElement, String aList, Object aFinder) {
        super(aMessage, anOperationName, anIndex,
              anElement, aList, aFinder);
    }

    public static ListEditInput newCase( OperationName anOperationName, int anIndex,
                                          Object anElement, String aList, Object aFinder) {
        String aMessage = toString(anOperationName, anIndex,
                                   anElement, aList);
        ListEditInput retVal = new ListEditInput(aMessage,
                                      anOperationName, anIndex,
                                      anElement, aList, aFinder);
    }
```
protected static void processInput(String anInput) {
    ListEditInput.newCase(OperationName.ADD, history.size(),
                          anInput, ApplicationTags.HISTORY,
                          MonolithicEchoer.class);
    String aFeedback = EchoUtilities.echo(anInput);
    addToHistory(aFeedback);
    ListEditMade.newCase(OperationName.ADD, history.size(),
                          anInput, ApplicationTags.HISTORY,
                          MonolithicEchoer.class);
    displayOutput(aFeedback);
    ListEditDisplayed.newCase(OperationName.ADD,
                                history.size(), anInput, ApplicationTags.HISTORY,
                                MonolithicEchoer.class);
}
Find all uses of a Method
How is testing done today?

LauncherOfMonolithicEchoTester

LauncherOfModularEchoTester
I/O Diff-Based Testing

Compare “correct” transcript with test transcript

- No creativity allowed in implementation defined I/O such as debugging statements and prompts
- Cannot distinguish between algorithms that have the same I/O behavior – e.g. monolithic and modular echo
- Thread scheduling and other non determinism can effect the trace of a correct algorithm
TRACe DIFFs

Compare “correct” trace with test trace

Can use filtering to test algorithms at multiple levels of abstractions and different aspects of algorithms

No formal description of algorithm – who checks the correct implementation
**Inter-Trace vs Intra-Trace**

Find relationships among steps within a trace

User input should be followed by a certain sequence of events which are different for different algorithms

The arguments of these events should have certain relationships

Other aspects of the trace such as source may have relationships
# Monolithic Specification and Testing

For each input I

I should be followed by ListEditInput, ListEditMade, and ListEditDisplayed

The operation name, index, element, and list should be the same in the events above

The element should be echo(I)
# Modular Specification and Testing

For each input `I`:

- The operation name, index, element, and list should be the same in the events above.
- The element should be `echo(I)`.
- The source of `ListEditNotified` and `ListEditObserved` should be different.
- The source of other objects can also be different as a model/interactor may be divided into multiple submodels/interactors.

A program that passes the modular tester will pass the monolithic tester.

Demoed OT algorithm was tested using traces.
SUMMARY

- Algorithm needed for understanding programs, testing, debugging
- Modularity increases need as steps scattered through many classes
- Important to be able to tie program to algorithm and keep them consistent
- Prints cannot be disabled and easily separated from real output,
- Debugging does not support race conditions and does not provide persistent tracing
- Untyped-event log can be turned off and on and filtered based on keyword, class, package
- Typed events allow algorithm steps to be in separate packages, filtering by event type, and ways to find implementations of a step
- Inter-trace diffs allow algorithm rather than I/O comparisons
- Intra-trace processing allows specification of algorithms and testing without correct traces.
NOT COVERED

- Types events can be listened through a message bus
- A message bus connects observers to observables
- Can block events when certain conditions are met
- Less heavyweight untyped traces also possible
- Following slides from previous class cover this
  - They duplicate some of the material here
TRACE

Please enter message:
The woods are lovely
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper:inputport
I*** (inputport.datacomm.simplex) Forwarding sent message java.nio.HeapByteBuffer
I*** (inputport.datacomm.simplex.buffer.nio) Sending message: java.nio.HeapByteBuffer
I*** (inputport.datacomm.simplex.buffer.nio) ABufferedWrite with id: 0 contents: java.
I*** (inputport.datacomm.simplex.buffer.nio) Started storing of buffered write with
I*** (inputport.datacomm.simplex.buffer.nio) Write op for: java.nio.channels.Selector select unblocks
I*** (inputport.datacomm.simplex.buffer.nio) Write op for: java.nio.channels.Selector select unblocks
I*** (inputport.datacomm.simplex) notifying to send listeners message: java.nio.HeapByteBuffer
Alice --> Echo Server: (0) java.nio.HeapByteBuffer [pos=0 lim=20 cap=20]
I*** (inputport.datacomm.simplex.buffer.nio) notified listeners about write
I*** (inputport.datacomm.simplex.buffer.nio) Selector registering read as no pending write
I*** (inputport.datacomm.simplex.buffer.nio) channel not connected or no pending write
I*** (inputport.datacomm.simplex.buffer.nio) Selector calls select

Tracer.showInfo(true);

May not want to to know about nio
Filtering: Selecting which events to print?
ONLY INPUTPORT.DATACOMM.SIMPLEX.BUFFER

AServerSimplexBufferInputPortLauncher [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 17, 2011 4:22:27)
I*** (inputport.datacomm.simplex.buffer) Retrieved from global state receive trapper.
I*** (inputport.datacomm.simplex.buffer) Changing connection status and asking driver.
I*** (inputport.datacomm.simplex.buffer) Received message: java.nio.HeapByteBuffer.
I*** (inputport.datacomm.simplex.buffer) Associating Alice with java.nio.channels.
I*** (inputport.datacomm.simplex.buffer) ServerInputPort connected to: java.nio.channels.

Echo Server<-->Alice (Opened)
I*** (inputport.datacomm.simplex.buffer) Received message: java.nio.HeapByteBuffer.
I*** (inputport.datacomm.simplex.buffer) ServerInputPort received message java.nio.ByteBuffer.

AnAliceSimplexBufferInputPortLauncher [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Oct 17, 2011 4:22:33)
I*** (inputport.datacomm.simplex.buffer) Retrieved from global state send trapper.
I*** (inputport.datacomm.simplex.buffer) Set my send trapper to: inputport.datacomm.
I*** (inputport.datacomm.simplex.buffer) Adding send listener: inputport.datacomm.
I*** (inputport.datacomm.simplex.buffer) Asking driver to connect and changing status.
I*** (inputport.datacomm.simplex.buffer) Received connected notification from driver.
I*** (inputport.datacomm.simplex.buffer) Sending to server my name: Alice.
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper: inputport.

Alice<-->Echo Server (Opened)
I*** (inputport.datacomm.simplex.buffer) Received sent notification from driver.
I*** (inputport.datacomm.simplex.buffer) Notifying to send listeners message: java.nio.HeapByteBuffer [pos=0 lim=5 cap=5].
Please enter message:
The woods are lovely.
I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper: inputport.
Please enter message:
I*** (inputport.datacomm.simplex.buffer) Received sent notification from driver.
I*** (inputport.datacomm.simplex.buffer) Notifying to send listeners message: java.nio.HeapByteBuffer [pos=0 lim=20 cap=20].
Alice<-->Echo Server: (1) java.nio.HeapByteBuffer [pos=0 lim=20 cap=20].
**Explicit Keywords**

```java
public static void info(String keyWord, String info);

Tracer.info("inputport.datacomm.simplex.buffer", "Asking driver to connect and changing status");

public static void setKeywordPrintStatus(
    String keyWord,
    Boolean status);

Tracer.setKeywordPrintStatus("inputport.datacomm.simplex.buffer", true)

Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
```

Have to specify package name each time

Package name can change
**Implicit KeyWords**

```java
public static void info(Object object, String info);

Tracer.info(this, "Asking driver to connect and changing status");

public static void setKeywordPrintStatus(Class c, Boolean status);

Tracer.setKeywordPrintStatus(
    AGenericSimplexBufferClientInputPort.class,
    false);
```
SHOWING PACKAGE NAMES

Cannot identify classes of the objects printing out messages

Need to not only specify which events to display but what information to display about each event

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(
        AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.PACKAGE_NAME);

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(
        AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.PACKAGE_NAME);

The woods are lovely

Please enter message:

I*** (inputport.datacomm.simplex.buffer) Forwarding message to send trapper:inputtp
Please enter message:

I*** (inputport.datacomm.simplex.buffer) Received sent notification from driver
I*** (inputport.datacomm.simplex.buffer) Notifying to send listeners message:java.
Alice-->Echo Server:(0) java.nio.HeapByteBuffer[pos=0 lim=5 cap=5]
SHOWING SHORT CLASS NAMES

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(
  AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME)

I*** (AGenericSimplexBufferClientInputPort) Received sent notification from driver
I*** (ASendRegistrarAndNotifier) Notifying to send listeners message: java.nio.HeapByteBuffer
Alice --> Echo Server: (0) java.nio.HeapByteBuffer [pos=0 lim=5 cap=5]
Please enter message:
The woods are lovely
I*** (AGenericSimplexBufferClientInputPort) Forwarding message to send trapper:inputpc:
Please enter message:
I*** (AGenericSimplexBufferClientInputPort) Received sent notification from driver
I*** (ASendRegistrarAndNotifier) Notifying to send listeners message: java.nio.HeapByteBuffer
Alice --> Echo Server: (1) java.nio.HeapByteBuffer [pos=0 lim=20 cap=20]
public static void setMessagePrefixKind(
    MessagePrefixKind newValue)

public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}
DISPLAYING ALL CLASSES IN PACKAGE

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setKeywordPrintStatus(
    AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME)

What if we want to focus on one class?

Narrow down which events
CONTROLLING IMPLICIT KEYWORD

Tracer.showInfo(true);
Tracer.setKeywordPrintStatus(Tracer.ALL_KEYWORDS, false);
Tracer.setImplicitKeywordKind(ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setKeywordPrintStatus(
    AGenericSimplexBufferClientInputPort.class, true);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME);
**Implicit Keyword**

```java
public static void setImplicitKeywordKind(
        ImplicitKeywordKind newValue)
```

```java
public enum ImplicitKeywordKind {
        OBJECT_TO_STRING,
        OBJECT_CLASS_NAME,
        OBJECT_PACKAGE_NAME
    }
```
TRACER STATIC METHODS SUMMARY

```java
setKeywordPrintStatus(String keyWord, Boolean status);

info(String keyWord, String info);

setImplicitKeywordKind(ImplicitKeywordKind newValue);

info(Object obj, String info);

public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}

setMessagePrefixKind(MessagePrefixKind newValue);

public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}
```
DEBUGGING CAPABILITIES IN TRACER?

- Blocking but separate windows for different processes
- See state at traced actions (with and without blocking)
- Separate state for different threads (with and without blocking)
- Have application-specific code learn about traced calls (perhaps in different processes)
setKeywordPrintStatus(String keyWord Boolean status);

info(String keyWord, String info);

setImplicitKeywordKind(ImplicitKeywordKind newValue);

info(Object obj, String info);

public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}

setMessagePrefixKind(MessagePrefixKind newValue);

public enum MessagePrefixKind {
    NONE,
    OBJECT_TO_STRING,
    SHORT_CLASS_NAME,
    FULL_CLASS_NAME,
    PACKAGE_NAME
}
@DisplayToString(true)
@ComponentWidth(1000)
public class MVCTraceableInfo extends TraceableInfo{
    public MVCTraceableInfo(String aMessage, Object anAnnouncer) {
        super(aMessage, anAnnouncer);
    }
    public static MVCTraceableInfo newInfo(String aMessage, Object aFinder) {
        MVCTraceableInfo retVal = new MVCTraceableInfo(aMessage, aFinder);
        retVal.announce();
        return retVal;
    }
}

Tracer.info(this, “MVC structure built”)
MVCTraceableInfo( “MVC structure built”, this);
ObjectEditor.edit(
TraceableDisplayAndWaitManagerFactory.
getTraceableDisplayAndPrintManager());

synchronized
waitForUser();

wait();

synchronized
proceed();

notify();
MESSAGE BUS

Like an observable it has registration method

Has an announce method

Does not generate events – simply communicates them to observers
RUN FUNCTIONALITY
public ADuplexRPCClientRelayingCollaborativeMVCLauncher(
    String aClientName, String aServerHost,
    String aServerId, String aServerName) {
    super(aClientName, aServerHost, aServerId, aServerName);
    ObjectEditor.edit
        (TraceableDisplayAndWaitManagerFactory.
            getTraceableDisplayAndPrintManager());
    Tracer.setKeywordDisplayStatus("this", true);
}
Each process can have separate trace window.
Each Trace Window has Separate Thread Area

Thread interacting with underlying communication channel

Thread invoking remote calls
THREAD DISPLAY

Thread name

Other info
Goals

Debugger allows programmer to set breakpoint, resume, and at each breakpoint, pause and see stack trace, variable values, and console message when some line of code is executed.

Tracer should provide equivalent

Set break point
BP Resume
BP Console message
BP Variable Values
CONSOLE OUTPUT EQUIVALENT: TYPED TRACING MESSAGE

GIPC-Defined Call Initiated Type

Programmer-defined MVCTraceable Type

Instance specific message

getMessage()

setMessage()

Traceable

announce()
STACK TRACE EQUIVALENT: STACK TRACE

[ATraceableDisplayAndWaitManager]

Common  ATraceableDisplayAndWaitManager

Selecting Thread

port.trace.CallInitiated: Queued: inputport.rpc.group.mvc.collaborative.relaying.example.Method

Asynchronous Received Call Invoker

inputport.rpc.group.mvc.collaborative.relaying.example.Method

1: Forward  Back  Proceed

2: Forward  Back  Proceed

CallInitiated
- Get Remote End Point
- Announce
- Get Call
- Get Message
- Init(Object)
- Print Stack Trace(PrintWriter)
- Get Wait

Get Stack Trace
- Get Time Stamp
- Print Stack Trace
- Init Cause(Throwable)
- Get Localized Message
- Set Stack Trace(StackTraceElement[])
- Print Stack Trace(PrintStream)
- Get Cause
- Get Display
- Get Finder
- Fill In Stack Trace
### Stack Trace Display

<table>
<thead>
<tr>
<th>Class Name</th>
<th>File Name</th>
<th>Line Number</th>
<th>Method Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>port.trace.CallInitiated</td>
<td>CallInitiated.java</td>
<td>21</td>
<td>newCase</td>
</tr>
<tr>
<td>input.rpc.simplex.ASimplexCallReceiveTrapp</td>
<td>ASimplexCallReceiveTrapp</td>
<td>38</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.rpc.duplicate.ADuplicateCallReceiveTrapp</td>
<td>ADuplicateCallReceiveTrapp</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.rpc.simplex.ASimplexRPCServerInput</td>
<td>ASimplexRPCServerInput</td>
<td>122</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.AREceiveRegistrarAndNotify</td>
<td>AReceiveRegistrarAndNotify</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.AREceiveMessageForward</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.group.AnAbstractGroupInputPort</td>
<td>AnAbstractGroupInputPort</td>
<td>243</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm.AREceiveRegistrarAndNotify</td>
<td>AReceiveRegistrarAndNotify</td>
<td>28</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm.AREceiveMessageForward</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm. AREceiveMessageForward</td>
<td>AReceiveMessageForward</td>
<td>20</td>
<td>notifyPortReceive</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>156</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm. AGenericSimplexBufferServer</td>
<td>AGenericSimplexBufferServer</td>
<td>87</td>
<td>messageReceived</td>
</tr>
<tr>
<td>input.datacomm. AGenericSimplexBufferServer</td>
<td>AGenericSimplexBufferServer</td>
<td>92</td>
<td>socketChannelRead</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>80</td>
<td>notifyRead</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>43</td>
<td>execute</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>45</td>
<td>processRead</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>180</td>
<td>processSelectedOperation</td>
</tr>
<tr>
<td>input.datacomm. AMultiLayerProtocolManager</td>
<td>AMultiLayerProtocolManager</td>
<td>207</td>
<td>run</td>
</tr>
</tbody>
</table>
INHERITANCE HIERARCHY

- `getMessage()`
- `setMessage()`
- `announce()`
- `getStackTrace()`
- `RuntimeException`
- `Traceable`
- `IS-A` relationship

**Diagram:**
- `Common` class
- `ATraceableDisplayAndWaitManager` class
- Methods:
  - `selectingThread()`
  - `asynchronousReceivedCallInvoker()`
  - `callInitiated()`
  - `getRemoteEndPoint()`
  - `announce()`
  - `getCall()`
  - `getMessage()`
  - `init(Object)`
  - `printStackTrace(PrintWriter)`
  - `getWait()`
  - `getStackTrace()`
  - `getTimeStamp()`
EXPLORING VARIABLES: EVENT PROPERTIES

Can see properties of event by calling getters.
Thread Has History of Typed Trace Info

Thread interacting with underlying communication channel.
BROWSING: LAST TO LAST CALL
RESUME: PROCEED BUTTON
SET BREAK POINT: PAUSE/DISPLAY ALL ANNOUNCED MESSAGES?

May want to pause only some of the announcements

May want to display only some of the announcements

How to specify a set of related announcements that should be displayed or paused?
**USING ANNOUNCER OBJECT ATTRIBUTES**

- `static setKeywordDisplayStatus(Object announcer, boolean status)`
- `static setImplicitDisplayKeywordKind(ImplicitKeywordKind val)`
- `static setImplicitWaitKeywordKind(ImplicitKeywordKind val)`

```java
public enum ImplicitKeywordKind {
    OBJECT_TO_STRING,
    OBJECT_CLASS_NAME,
    OBJECT_PACKAGE_NAME
}
```

- `setKeywordDisplay(Wait)Status(announcer, true/false)` says that if an event is announced by an object whose `toString()`/class/package attribute is that of the announcer then it should be displayed(wait).

- `setImplicitDisplay(Wait)KeywordKind` determines if `toString()`, class or package attribute is used for all events.

Print, Display and Wait are three different things you can do with traced information.
Using Event Object

- getMessage
- setMessage
- announce
- setWait
- setDisplay

Traceable

IS-A

MVCTraceable
Debugger Issues Resolved

- Debugger makes it difficult to test race conditions
- All threads and processes mapped to a single code window
- Cannot see the history of actions taken by a thread
- Break points do not transfer to another computer
- Cannot use a mechanism to set multiple related debug points
**Event Class Filtering**

Tracer.showInfo(true);
Tracer.setImplicitPrintKeywordKind(ImplicitKeywordKind.OBJECT_CLASS_NAME);
Tracer.setMessagePrefixKind(MessagePrefixKind.SHORT_CLASS_NAME);
TraceableInfo.setPrintSource(true);
TraceableInfo.setPrintTime(true);
TraceableInfo.setPrintThread(true);
Tracer.setKeywordPrintStatus(ListEditMade.class, true);
Tracer.setKeywordPrintStatus(ListEditObserved.class, true);

Two sources, as the same event fired twice, first by the real source and then by the event object pretending to be the source.

All events of type ListEditMade or ListEditObserved.

Can be announced by different sources.

Alternative (source/event) class-based filtering?