monitoring with AspectJ

CS 119

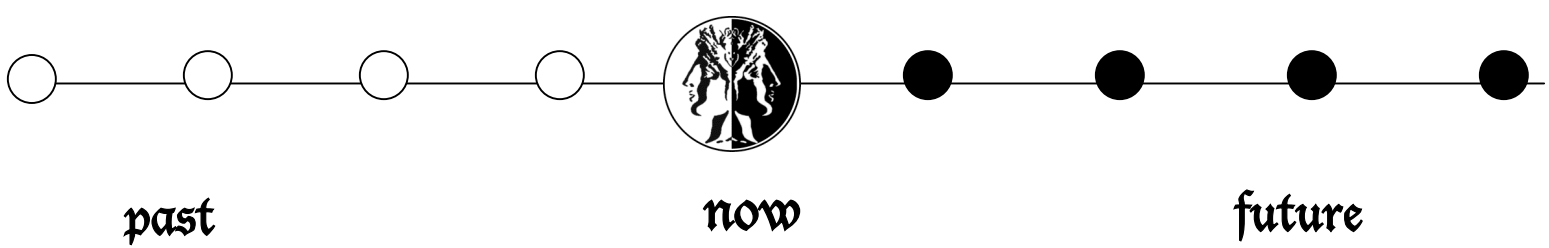
examples illustrating
how to write monitors

Overview

- this and previous lecture should enable you to write monitors in AspectJ
- we will see examples of AspectJ monitors
- how to structure monitors
- trace view of program executions
  - reference to the past
  - obligations for the future
  - state machines
- giving an idea of how it is done without the use of special specification notation
- preparing for later lectures on temporal logic

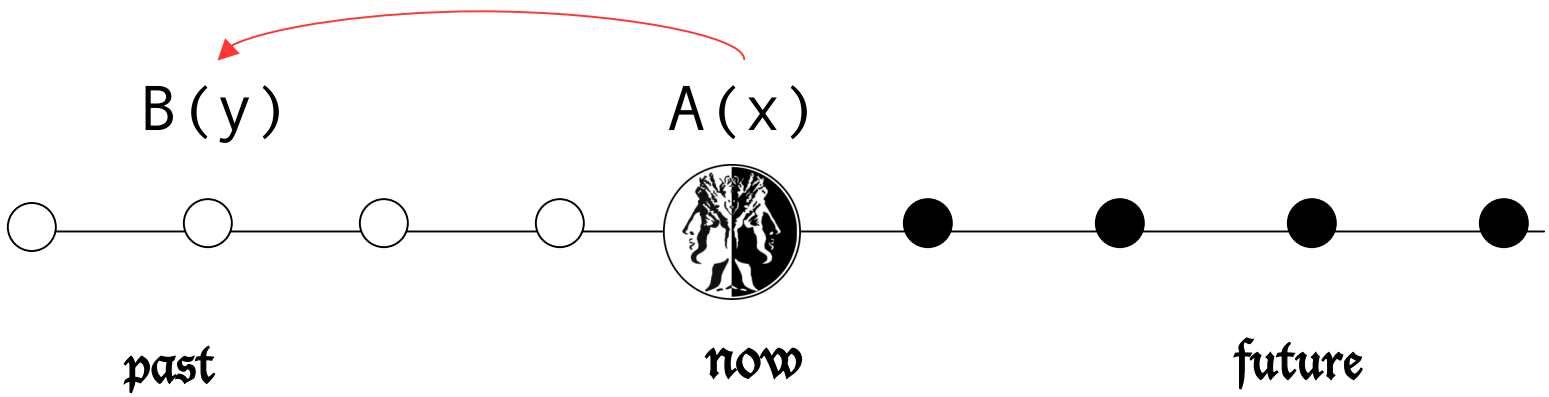
recall trace view of execution

- a formal view of an execution is to consider it as a sequence σ of program states: $\sigma = s_1\ s_2\ s_3\ \ldots\ s_n$
- during program execution we are at any point in time in the present moment now where the past is known but the future is not known.



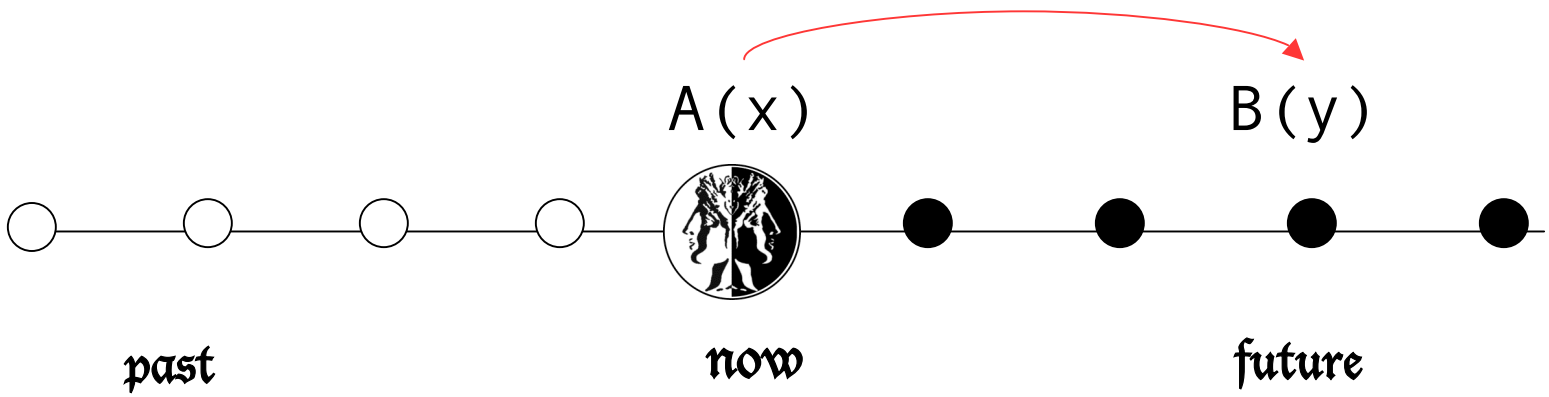
past time properties

- If A(x) happens now then B(y) must have happened in the past where R(x,y)



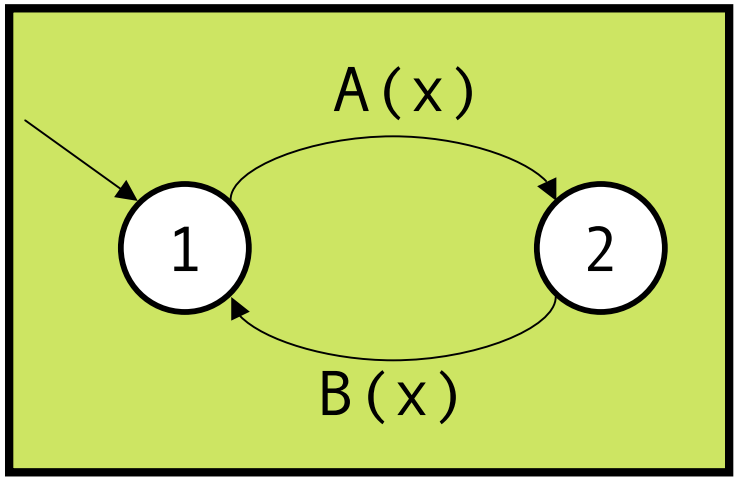
future time properties

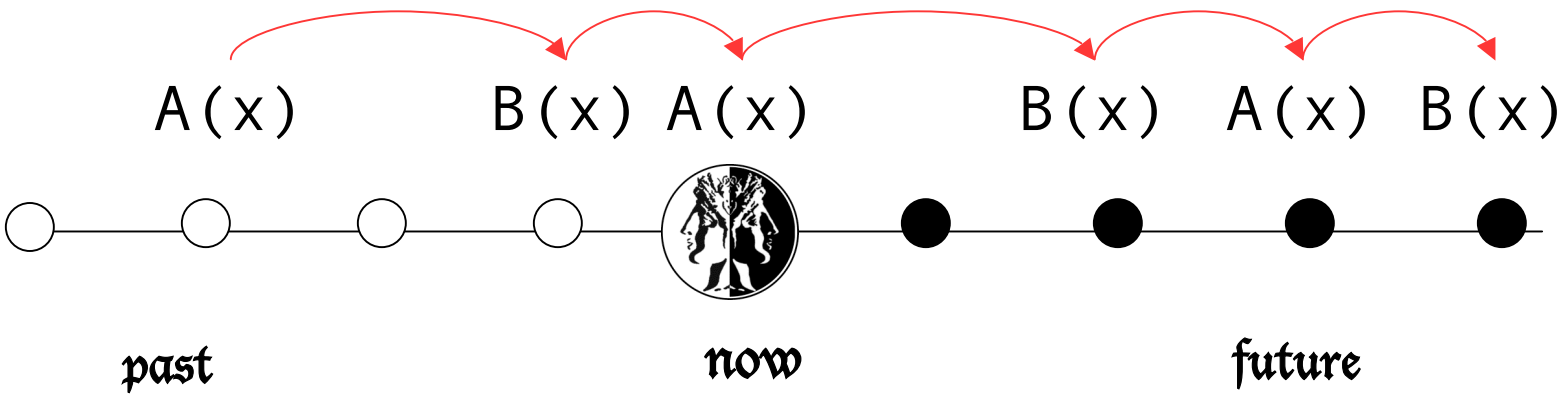
- If A(x) happens now then B(y) must happen in the future where R(x,y)



state machines

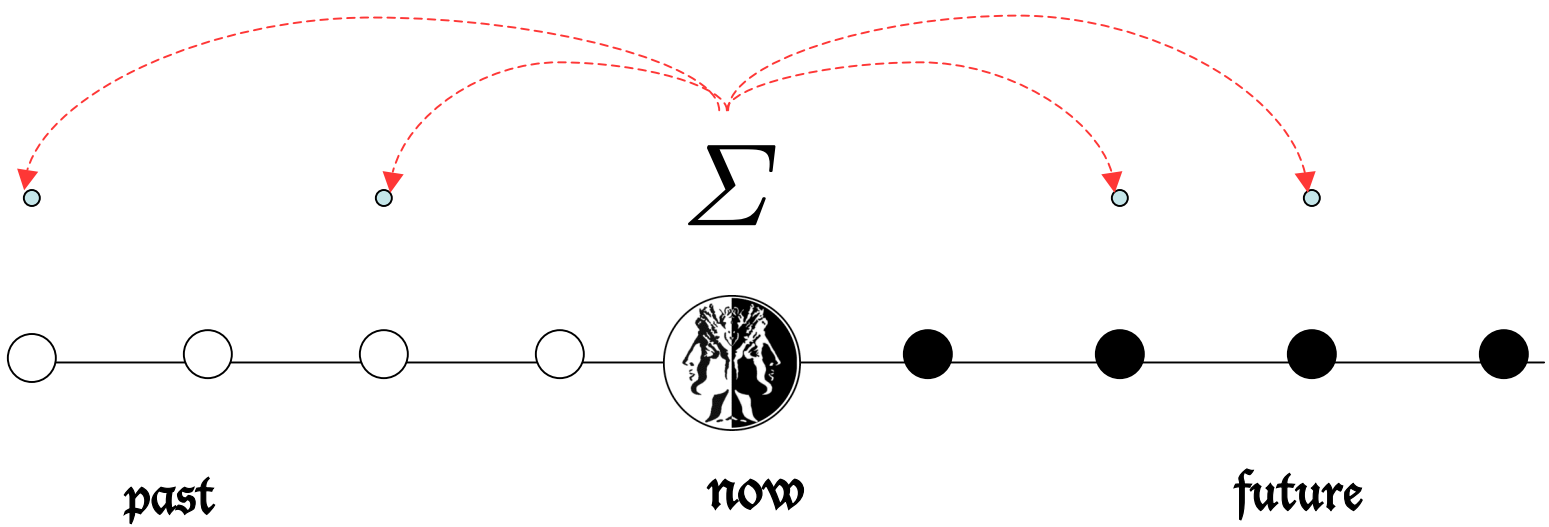
- A(x) and B(x) should happen in an alternating manner





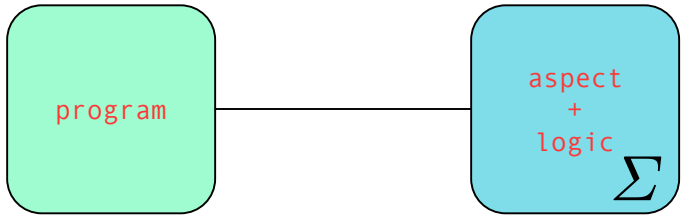
monitor keeps state Σ

Σ = past events $\times$ future obligations

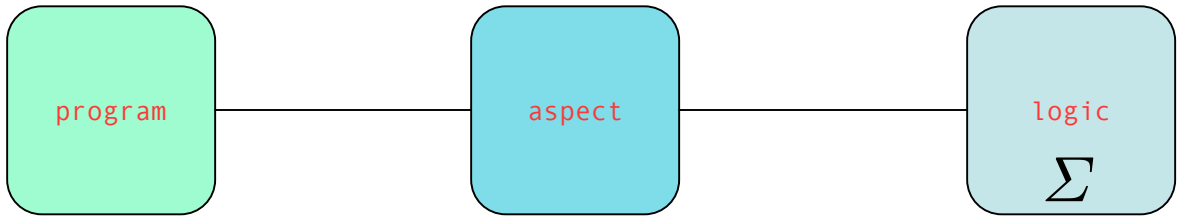


two monitoring architectures

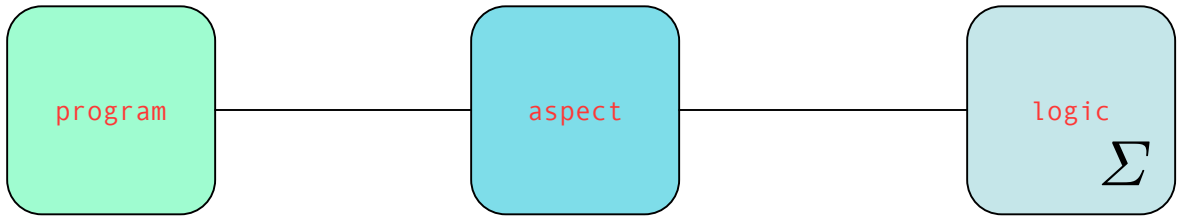
- write checking logic inside aspects



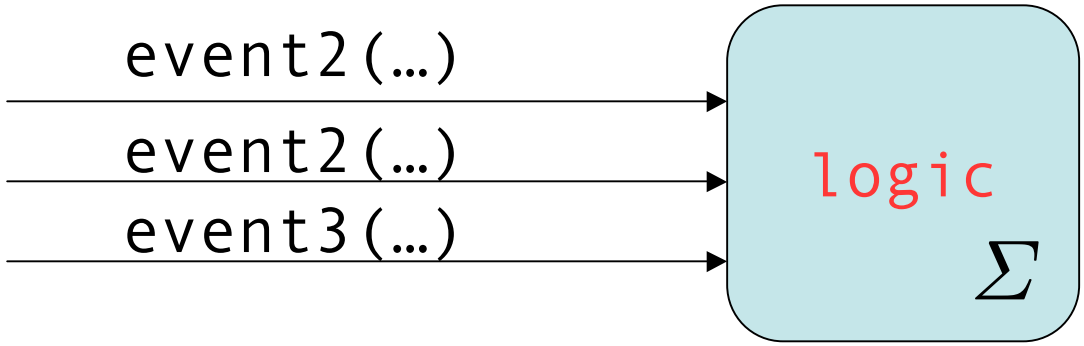
- write checking logic in separate class



logic aside

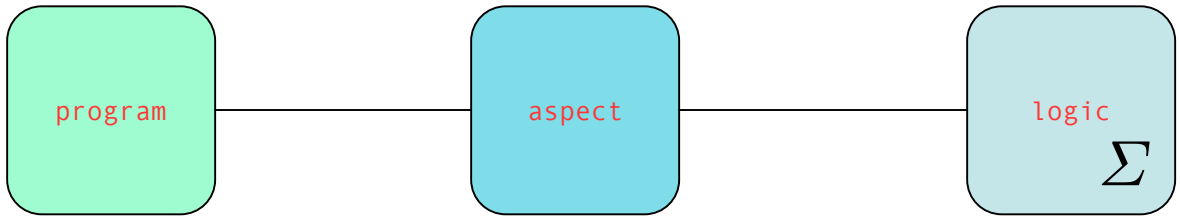


- separating logic into a separate module makes specification writing modular.

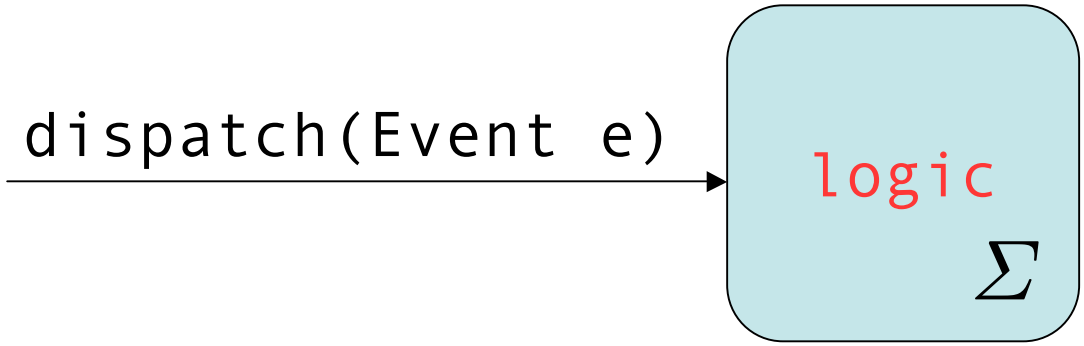


either: a method for each kind of event

logic aside



- separating logic into a separate module makes specification writing modular.



or: a single event dispatch method

a file example

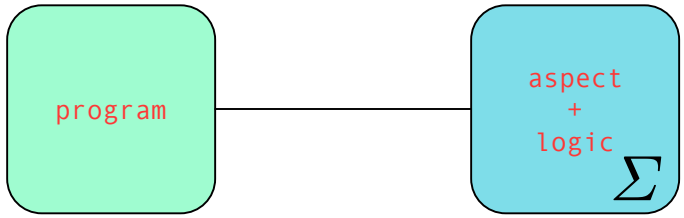
```
class File {
    public static final int READ  = 1;
    public static final int WRITE = 2;

    public File(String name, int mode){…}

    public void write(String text){…};
    public String read(){…};
    public void close() {…}
    public String getName() {…}
}
```

a requirement

A file should be accessed according to its mode READ or WRITE. A file cannot be accessed or closed unless it has been opened.

```
File file = new File("data",File.WRITE);
file.write("monitor");
file.write("this if you can");
file.write(file.read());          ← error
file.close();
file.close();                     ← error
```

first: all code in one aspect



$$\Sigma = \text{File} \rightarrow \{\underset{1}{\text{READ}},\underset{2}{\text{WRITE}}\}$$

$$\begin{bmatrix} \text{file}_1 \rightarrow 1 \\ \text{file}_2 \rightarrow 2 \\ \text{file}_3 \rightarrow 1 \\ \text{file}_4 \rightarrow \text{null} \\ \ldots \end{bmatrix}$$

null : closed

Identity

```
IdentityHashMap<File,Integer> modes = new IdenityHashMap();
```

Identity Maps in Java

- a normal hashmap does not work:
  - uses `equals` method to determine equality
  - and file objects may change wrt. equals

```
HashMap<File,Integer> modes = new HashMap();
```

- necesarry to use identity hashmap:
  - uses `==` to determine file equality

```
IdentityHashMap<File,Integer> modes = new IdentityHashMap();
```

all in one aspect

```
public aspect FileMonitor {
    pointcut open(int mode)  : call(File.new(String,int)) && args(..,mode);
    pointcut write(File file): call(void File.write(..)) && target(file);
    pointcut read(File file) : call(String File.read(..)) && target(file);
    pointcut close(File file): call(void File.close()) && target(file);
    …
    IdentityHashMap<File,Integer> modes = new IdentityHashMap();

    after(int mode) returning (File file) : open(mode) {
      modes.put(file,mode);
    }

    before(File file) : write(file) {
        Integer mode = modes.get(file);
        if(mode == null || mode != File.WRITE)
            error("illegal write to file " + file.getName());
    }
    before(File file) : read(file) {
        Integer mode = modes.get(file);
        if(mode == null || mode != File.READ)
            error("illegal read from file " + file.getName());
    }
    before(File file) : close(file) {
        if (modes.get(file) == null)
            error("attempt to close closed file " + file.getName());
        modes.remove(file);
    }
}
```

a simple way of getting an informative error message

```
void error(String str) {
    try {
      throw new Exception ("*** " + str);
    }
    catch (Exception e) {
       e.printStackTrace();
    }
}
```

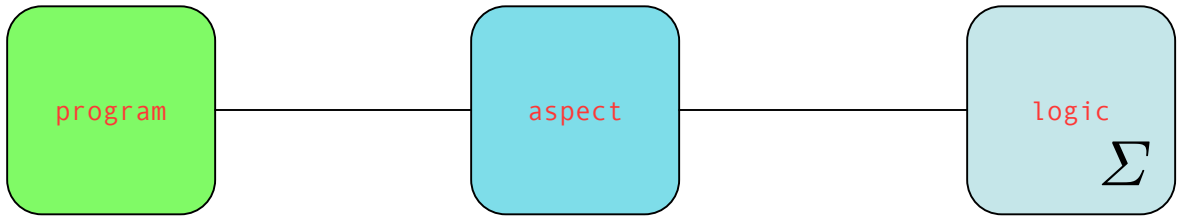
error message in Eclipse

Java - cs119mon/src/aspectj2/architecture/Test.java - Eclipse SDK - /Users/khavelun/Desktop/development/workspace

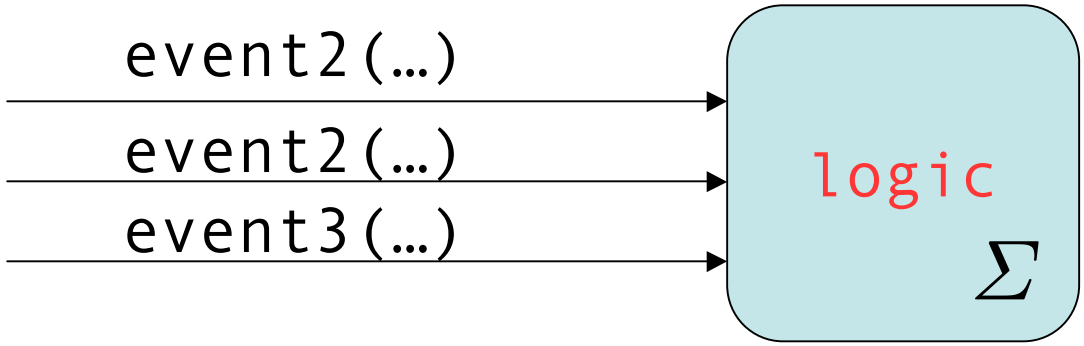
```
1  package aspectj2.architecture;
2
3  public class Test {
4      public static void main(String[] args){
5          File file = new File("data",File.WRITE);
6          file.write("monitor");
7          file.write("this if you can");
8          file.write(file.read());
9          file.close();
10         file.close();
11     }
12 }
13
```

```
<terminated> Test (9) [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (Apr 13, 2008 8:19:20 AM)

java.lang.Exception: *** illegal read from file data
    at aspectj2.architecture.FileMonitor1.error(FileMonitor1.aj:16)
    at aspectj2.architecture.FileMonitor1.ajc$before$aspectj2_architecture_FileMonitor1$3$e8703177(FileMonitor1.aj
    at aspectj2.architecture.Test.main(Test.java:8)

java.lang.Exception: *** attempt to close closed file data
    at aspectj2.architecture.FileMonitor1.error(FileMonitor1.aj:16)
    at aspectj2.architecture.FileMonitor1.ajc$before$aspectj2_architecture_FileMonitor1$4$cfff8209(FileMonitor1.aj:4
    at aspectj2.architecture.Test.main(Test.java:10)
```

logic aside



- separating logic into a separate module makes specification writing modular.



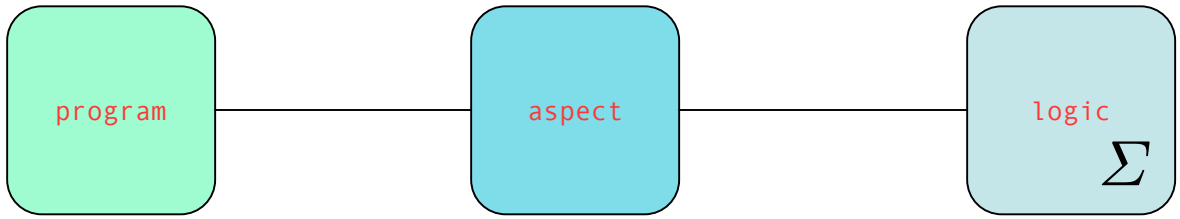
either: a method for each kind of event

```
class EngineM {
    IdentityHashMap<File,Integer> modes = new IdentityHashMap();
    …
    void open(File file, int mode) {
        modes.put(file,mode);
    }

    void write(File file) {
        Integer mode = modes.get(file);
            if(mode == null || mode != File.WRITE) {
                error("illegal write to file " + file.getName());
        }
    }

    void read(File file) {
        Integer mode = modes.get(file);
        if(mode == null || mode != File.READ) {
           error("illegal read from file " + file.getName());
        }
    }

    void close(File file) {
        if (modes.get(file) == null)
            error("attempt to close closed file " + file.getName());
        modes.remove(file);
    }
}
```

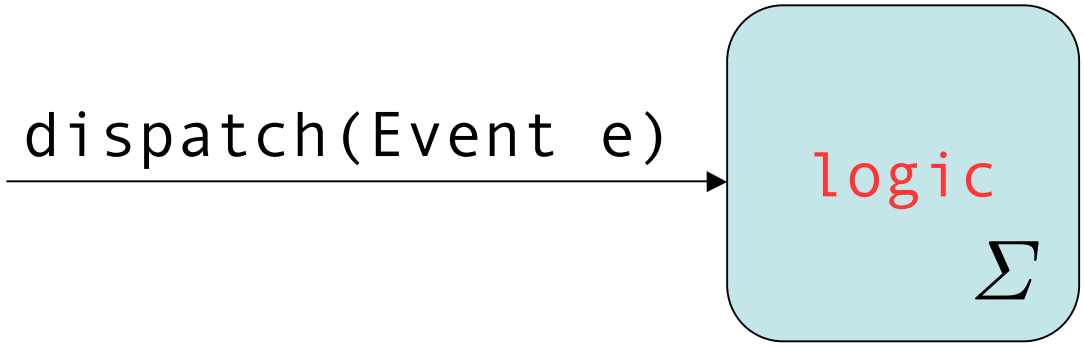
aspect calls logic

```
public aspect FileMonitor {
    pointcut open(int mode)  : call(File.new(String,int)) && args(..,mode);
    pointcut write(File file): call(void File.write(..)) && target(file);
    pointcut read(File file) : call(String File.read(..)) && target(file);
    pointcut close(File file): call(void File.close()) && target(file);

    EngineM engine = new EngineM();

    after(int mode) returning (File file) : open(mode) {
        engine.open(file,mode);
    }

    before(File file) : write(file) {
        engine.write(file);
    }

    before(File file) : read(file) {
        engine.read(file);
    }

    before(File file) : close(file) {
        engine.close(file);
    }
}
```

logic aside



- separating logic into a separate module makes specification writing modular.



or: a single event dispatch method

one dispatch function

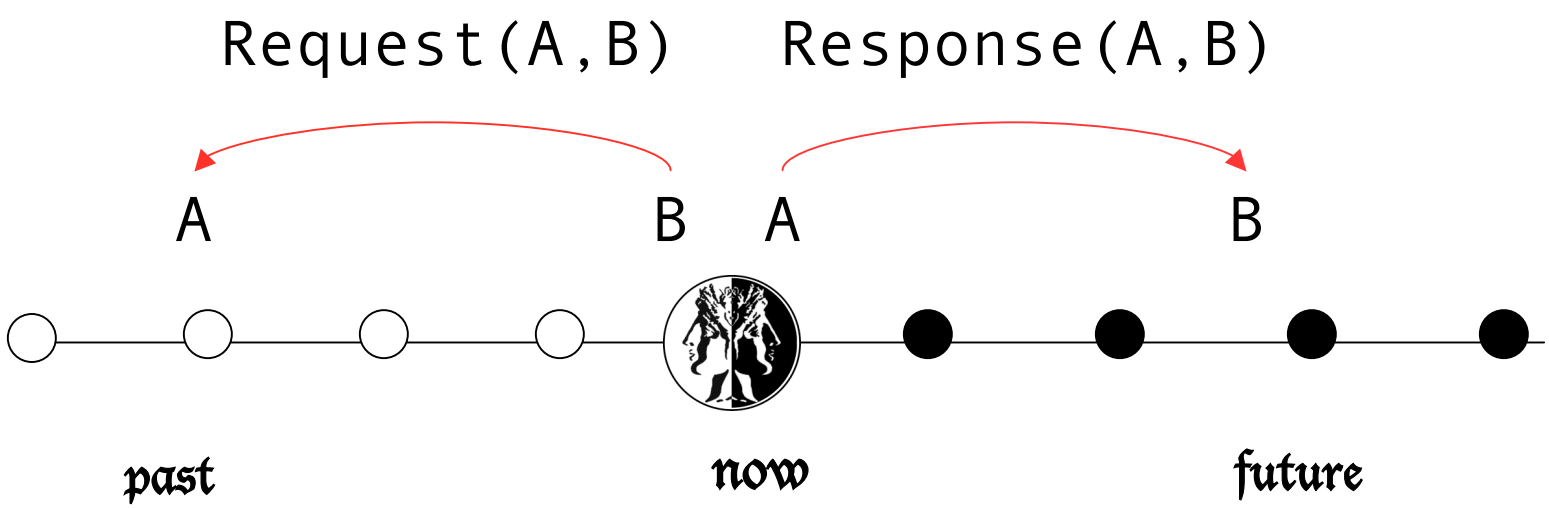
```
class EngineE {
    void dispatch(Event event) { … }
    …
}
```

```
class OpenEvent implements Event {
    public File file;
    public int mode;

    public OpenEvent(File file, int mode) {
        this.file = file;
        this.mode = mode;
    }
}
```

```
aspect FileMonitor {
    pointcut open(int mode)   : …;
    pointcut write(File file) : …;
    pointcut read(File file)  : …;
    pointcut close(File file) : …;

    EngineE engine = new EngineE();

    after(int mode) returning (File file) : open(mode) {
        engine.dispatch(new OpenEvent(file,mode));
    }

    …
}
```

parameterized temporal operators

– Define two temporal operators:
  - Response(o.A,o.B) : whenever method A is called on object o, eventually method B will be called on o.
  - Request(o.A,o.B) : whenever method B is called on o, in the past method A must have been called on o.



file example again

R1: A file should eventually be closed once opened.

R2: A file cannot be closed unless it has been opened.

```
File file1 = new File("data1",File.WRITE);
file1.open();
file1.write("monitor");
File file2 = new File("data1",File.WRITE);
file2.close();                    error
terminate();                      error
```

assume existence of a method: File.open()

abstract Response property

```
abstract aspect Response {
    abstract pointcut firstMethod(Object o);
    abstract pointcut secondMethod(Object o);
    abstract pointcut shutDown();

    IdentityHashSet obligations = new IdentityHashSet();

    before(Object o) : firstMethod(o) {
        obligations.add(o);
    }

    before(Object o) : secondMethod(o) {
        obligations.remove(o);
    }

    before() : shutDown() {
        Iterator it = obligations.iterator();
        while(it.hasNext()) {
            error("Matching closing method not found on object: " + it.next());
        }
    }
}
```

abstract
Request property

```
abstract aspect Request {
    abstract pointcut firstMethod(Object o);
    abstract pointcut secondMethod(Object o);

    IdentityHashSet history = new IdentityHashSet();

    before(Object o) : firstMethod(o) {
        history.add(o);
    }

    before(Object o) : secondMethod(o) {
        if (!history.contains(o)) {
            error("Matching opening method not found on object " + o);
        } else
            history.remove(o);
        }
    }
}
```

concrete Response and Request properties R1 and R2

```
aspect R1 extends Response {
    pointcut firstMethod(Object o) :
        call(void File.open()) && target(o);
    pointcut secondMethod(Object o) :
        call(void File.close()) && target(o);
    pointcut shutDown() :
        call(void Test.terminate());
}
```

```
aspect R2 extends Request {
    pointcut firstMethod(Object o) :
        call(void File.open()) && target(o);
    pointcut secondMethod(Object o) :
        call(void File.close()) && target(o);
    pointcut shutDown() :
        call(void Test.terminate());
}
```

Java - cs119mon/src/aspectj2/temporal/Test.java - Eclipse SDK - /Users/khavelun/Desktop/development/workspace

Package E | Hierarchy

- cs119mon
  - src
    - (default package)
    - aspectj1
    - aspectj2.any_hashset.we
    - aspectj2.architecture
    - aspectj2.coll_hashset.na
    - aspectj2.coll_hashset.we
    - aspectj2.locking
    - aspectj2.safeenum
    - aspectj2.statemachine
    - aspectj2.temporal
      - File.java
      - R1.aj
      - R2.aj
      - Request.aj
      - Response.aj
      - Test.java
    - aspectj2.timing
    - commons
    - old.future
    - old.past
  - JRE System Library [JVM 1.5.0
  - AspectJ Runtime Library
  - Referenced Libraries
  - doc
- Hello World
- k9
- lsccs_python
- >lsccs-jpl [porter.jpl.nasa.gov]
- >msproto1 [porter.jpl.nasa.gov]
- oldlsccsdsl
- pax
- Testbed
- testpydev
- testpylint
- threads

Test.java | File.java | R1.aj | R2.aj | Request.aj | Response.aj

```
3  public class Test {
4      static void terminate() {}
5
6      public static void main(String[] args){
7          File file1 = new File("data1",File.WRITE);
8          file1.open();
9          file1.write("monitor");
10         File file2 = new File("data1",File.WRITE);
11         file2.close();
12         terminate();
13     }
14 }
15
```

Problems | Javadoc | Declaration | Search | Console

<terminated> Test (4) [Java Application] /System/Library/Frameworks/JavaVM.framework/Versions/1.5.0/Home/bin/java (Apr 13, 2008 3:09:24 PM)

```
java.lang.Exception: *** Matching opening method not found on object data1
        at aspectj2.temporal.Request.error(Request.aj:15)
        at aspectj2.temporal.Request.ajc$before$aspectj2_temporal_Request$2$b839b92d(Request.aj:29)
        at aspectj2.temporal.Test.main(Test.java:11)

java.lang.Exception: *** Matching closing method not f[illegible] on object: data1
        at aspectj2.temporal.Response.error(Response.aj:15)
        at aspectj2.temporal.Response.ajc$before$aspectj2_temporal_Response$3$e564b9f7(Response.aj:34)
        at aspectj2.temporal.Test.main(Test.java:11)
```

Writable | Smart Insert | 11 : 23

timing properties

- record milliseconds used with
  - System.currentTimeMillis() : *"the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC."*
  - Requires a new event to occur
- use a "real" Timer object that spawns a thread which times out by itself. This does not require a new event to trigger. For example javax.swing.Timer.
- timestamp events in some other way in case they come from "outside" the application.

abstract timed Response property

```
abstract aspect TimedResponse {
    abstract pointcut firstMethod(Object o);
    abstract pointcut secondMethod(Object o);
    abstract pointcut shutDown();
    abstract boolean  timeok(long time);

    IdentityHashMap obligations = new IdentityHashMap();

    before(Object o) : firstMethod(o) {
        obligations.put(o,System.currentTimeMillis());
    }

    before(Object o) : secondMethod(o) {
        long secondTime = System.currentTimeMillis();
        Long firstTime = (Long)obligations.get(o);
        if (firstTime != null) {
            long time = secondTime - firstTime;
            if (!timeok(time))
                error("time " + time + " violates time constraint for " + o);
            obligations.remove(o);
        }
    }

    before() : shutDown() {… check for emptiness as before …}
}
```

back to the file example

TR1: After a file has been opened it should be closed within 5 seconds.

$\Box$(o.firstMethod $\rightarrow$ $\Diamond_5$ o.secondMethod)

```
aspect TR1 extends TimedResponse {
    pointcut firstMethod(Object o) :
        call(void File.open()) && target(o);
    pointcut secondMethod(Object o) :
        call(void File.close()) && target(o);
    pointcut shutDown() :
        call(void Test.terminate());

    boolean timeok(long time) {
        return time <= 5000;
    }
}
```

monitoring best programming practices

- generic properties we would want to hold for any program
- often concerns the use of various data types
- such properties are no different than domain specific properties
- some can be detected with static analysis
- however a dynamic analysis is relatively easy to program, whereas a static analysis either would not be possible or would require a substantial amount of work to implement

properties of Java library APIs

Iterator (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Joergen Ingman | TimeLife.com | Micro Site | Eventseer.net - Home | Free Website Polls | Richest Cou...n the World | Castle for Sale | Programming with C/C++ | Costa Rica, ...Estate Blogs | BASH Help - ...sh Tutorial

Try JavaMOP Online - FSL | Iterator (Java 2 Platform...

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages
- java.applet
- java.awt
- java.awt.color
- java.awt.datatransfer
- java.awt.dnd
- java.awt.event
- java.awt.font

IOException
IOR
IORHelper
IORHolder
IORInfo
IORInfoOperations
IORInterceptor
IORInterceptor_3_0
IORInterceptor_3_0Helper
IORInterceptor_3_0Holder
IORInterceptor_3_0Operations
IORInterceptorOperations
IRObject
IRObjectOperations
IstringHelper
ItemEvent
ItemListener
ItemSelectable
Iterable
Iterator
IvParameterSpec
JApplet
JarEntry
JarException
JarFile
JarInputStream
JarOutputStream
JarURLConnection
JButton
JCheckBox
JCheckBoxMenuItem
JColorChooser
JComboBox
JComboBox.KeySelectionManager

Overview | Package | **Class** | Use | Tree | Deprecated | Index | Help

PREV CLASS | NEXT CLASS | FRAMES | NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD — DETAIL: FIELD | CONSTR | METHOD

Java™ 2 Platform Standard Ed. 5.0

java.util

Interface Iterator<E>

All Known Subinterfaces:
ListIterator<E>

All Known Implementing Classes:
BeanContextSupport.BCSIterator, Scanner

public interface **Iterator<E>**

An iterator over a collection. Iterator takes the place of Enumeration in the Java collections framework. Iterators differ from enumerations in two ways:

- Iterators allow the caller to remove elements from the underlying collection during the iteration with well-defined semantics.
- Method names have been improved.

This interface is a member of the Java Collections Framework.

Since:
1.2

See Also:
Collection, ListIterator, Enumeration

Method Summary	
boolean	hasNext() Returns true if the iteration has more elements.
E	next() Returns the next element in the iteration.
void	remove() Removes from the underlying collection the last element returned by the iterator (optional operation).

Method Detail

hasNext

R_1: There should be no two calls to next() without a call to hasNext() in between, on the same iterator.

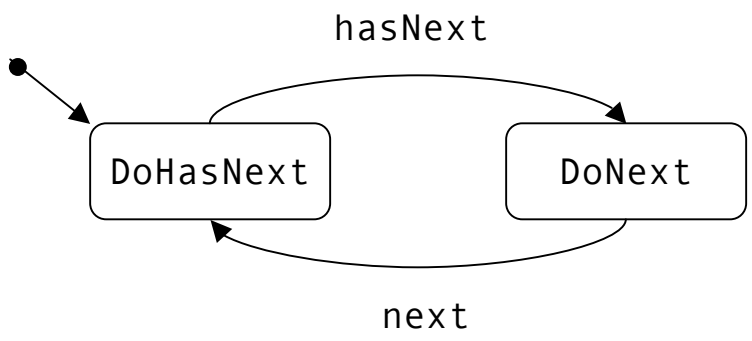
use of iterators

There should be no two calls to Iterator.next() without a call to Iterator.hasNext() in between, on same iterator

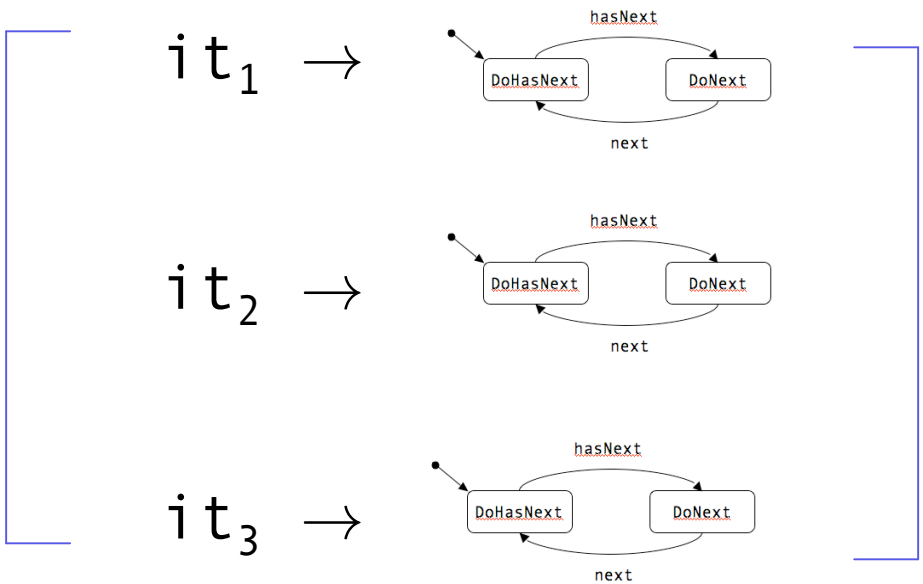
```
Vector<String> words = new Vector();

readWords(words);

Iterator it = words.iterator();

while(it.hasNext()) {
    String w1 = (String)it.next();
    String w2 = (String)it.next();
    storeCorrespondence(w1,w2);
    if (it.hasNext())
        System.out.println("there is more!");
}
```

error

slightly stronger property expressed as a state machine



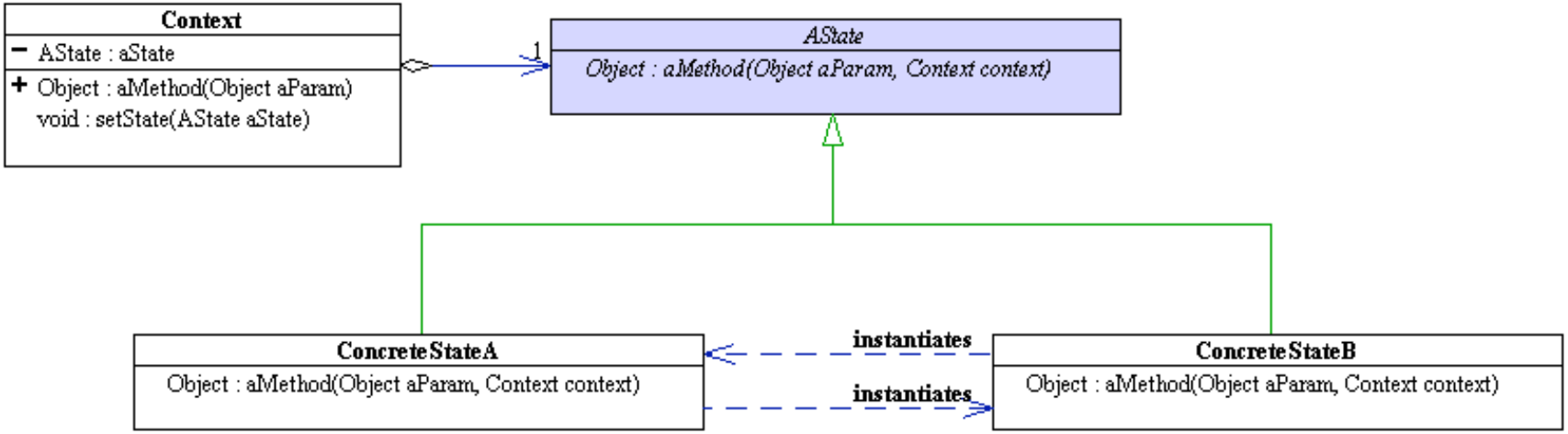
we need a state machine per iterator



the state design pattern

- Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.
- An object-oriented state machine

Gamma, Erich; Richard Helm, Ralph Johnson, John M. Vlissides (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley

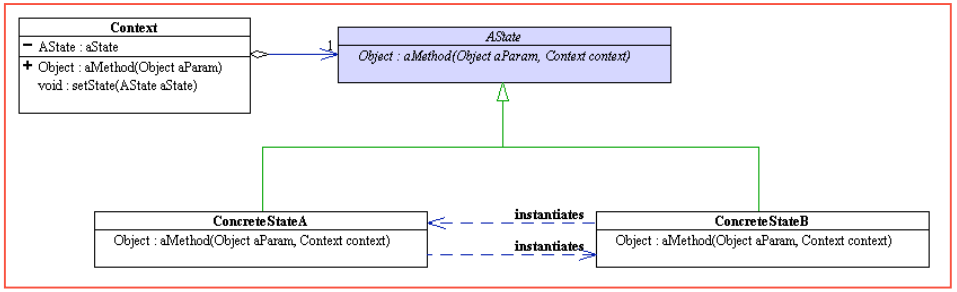


the state design pattern spelled out

- define a "context" class to present a single interface to the outside world.
- define a State abstract base class.
- represent the different "states" of the state machine as derived classes of the State base class.
- define state-specific behavior in the appropriate State derived classes.
- maintain a pointer to the current "state" in the "context" class.
- to change the state of the state machine, change the current "state" pointer.

the state machines

```
class Machine {
    State state = State.doHasNext;
    void hasNext() {
        state = state.hasNext();
    }
    void next() {
        state = state.next();
    }
}
```




```
class State {
    static final State doNext = new DoNext();
    static final State doHasNext = new DoHasNext();

    State hasNext(){
        System.out.println("*** warning: hasNext called unnecessarily");
        return this;
    }

    State next(){
        System.out.println("*** error: next called illegally");
        return this;
    }
}
```

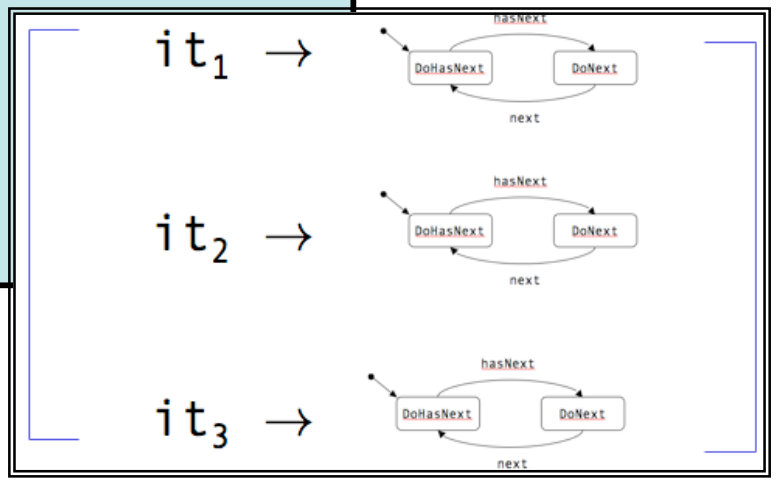
```
class DoHasNext extends State {
  State hasNext() {
      return doNext;
  }
}
```

```
class DoNext extends State {
    State next() {
        return doHasNext;
    }
}
```

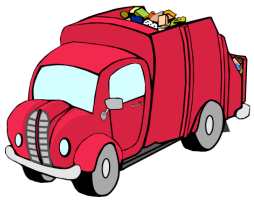
the aspect

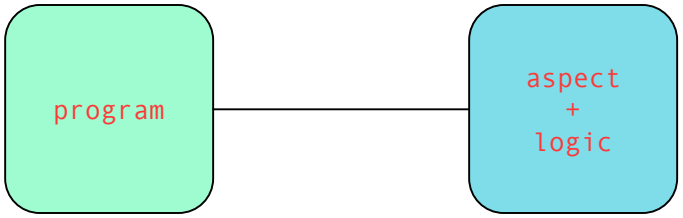
```
aspect HasNextPolicy {
    WeakIdentityHashMap monitors = new WeakIdentityHashMap();

    pointcut createiter():
        call(* java.util.Collection+.iterator());
    pointcut hasNext(Iterator it):
        call(* java.util.Iterator+.hasNext()) && target(it);
    pointcut next(Iterator it):
        call(* java.util.Iterator+.next()) && target(it);

    after() returning (Iterator it): createiter() {
        monitors.put(it,new Machine());
    }

    before(Iterator it): hasNext(it) {
       ((Machine)monitors.get(it)).hasNext();
    }

    before(Iterator it): next(it) {
        ((Machine)monitors.get(it)).next();
    }
}
```

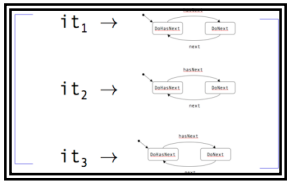
weak & identity



weak identity hash maps







- hasmap still an identity hashmap (using `==‘)
- a normal (identity) hashmap keeps a mapping until it is explicitly deleted with `Map.remove()’.
- this becomes a problem since the monitor will then accumulate bindings between iterators and state machines. The garbage collector cannot collect the iterators when they are no longer used by the monitored application.
- a weak collection releases an object to the garbage collector when it is no longer used by any other part of the program.

properties of Java library APIs

Enumeration (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages

java.applet

Entity
EntityReference
EntityResolver
EntityResolver2
Enum
EnumConstantNotPrese
EnumControl
EnumControl.Type
Enumeration
EnumMap
EnumSet
EnumSyntax
Environment
EOFException
Error
ErrorHandler
ErrorListener
ErrorManager
EtchedBorder
Event

Overview Package Class Use Tree Deprecated Index Help

Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

java.util

Interface Enumeration<E>

All Known Subinterfaces:

NamingEnumeration<T>

All Known Implementing Classes:

StringTokenizer

R_2: An enumeration should not be propagated after the underlying vector has been changed.

Method Summary

boolean	hasMoreElements() Tests if this enumeration contains more elements.
E	nextElement() Returns the next element of this enumeration if this enumeration object has at least one more element to provide.

Find: 97 Next Previous Highlight all Match case

Downloads Clear

enumerators faster than iterators but less safe

"I'd tried using Iterator and Enumeration to compare their performance on a Vector object containing 100 000 Strings. Enumeration was consistently about 50% faster". - *web blog*

An Iterator next() operation throws a:

ConcurrentModificationException

if it detects that the underlying collection has been modified while iteration is underway.

Enumerator does not!

use of enumerators

An enumeration should not be propagated after the undelying vector has been changed.

```
Vector v = new Vector();

v.add(1);
v.add(2);
v.add(3);

Enumeration en = v.elements();

while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```

error

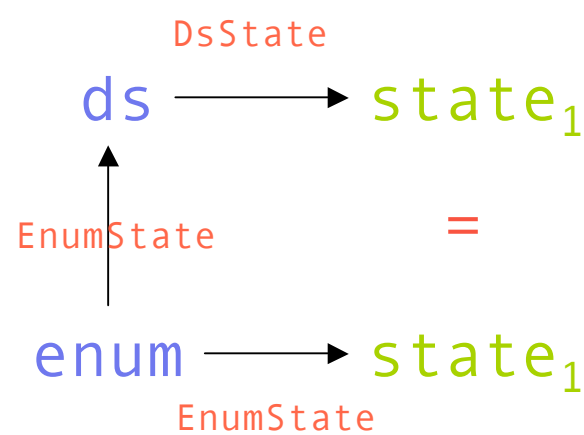
produces:

1
3
4

three maps are needed

- DsState: recording when a data structure was last updated (maps to unique object)
- EnumState: recording the state of the data structure of an enumeration at creation time
- EnumDs: recording what data structure corresponds to what enumeration

$$\begin{aligned} \mathtt{DsState} &= \mathtt{Ds} \rightarrow \mathtt{State} \\ \mathtt{EnumState} &= \mathtt{Enum} \rightarrow \mathtt{State} \\ \mathtt{EnumDs} &= \mathtt{Enum} \rightarrow \mathtt{Ds} \end{aligned}$$



example monitored run

```
…
v.add(3);
Enumeration en = v.elements();
while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```

example monitored run

```
…
v.add(3);
Enumeration en = v.elements();
while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```

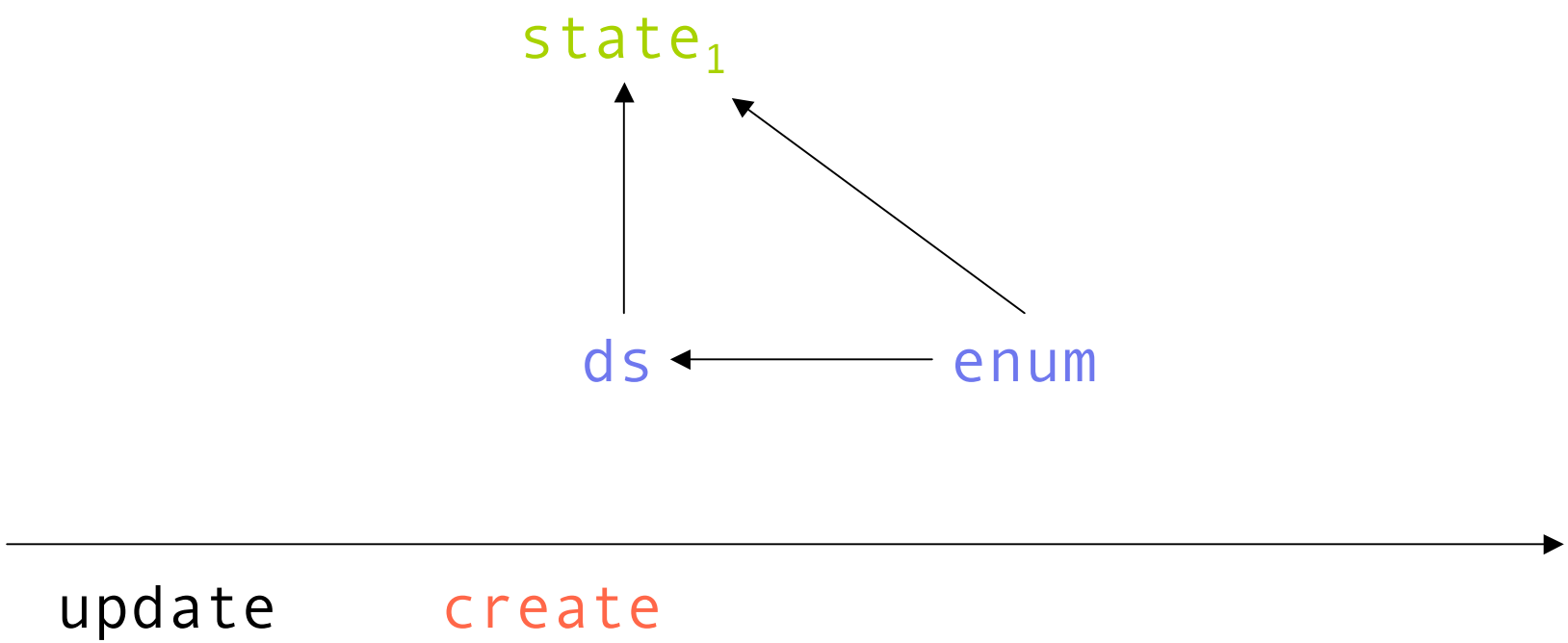
$state_1$

ds

update

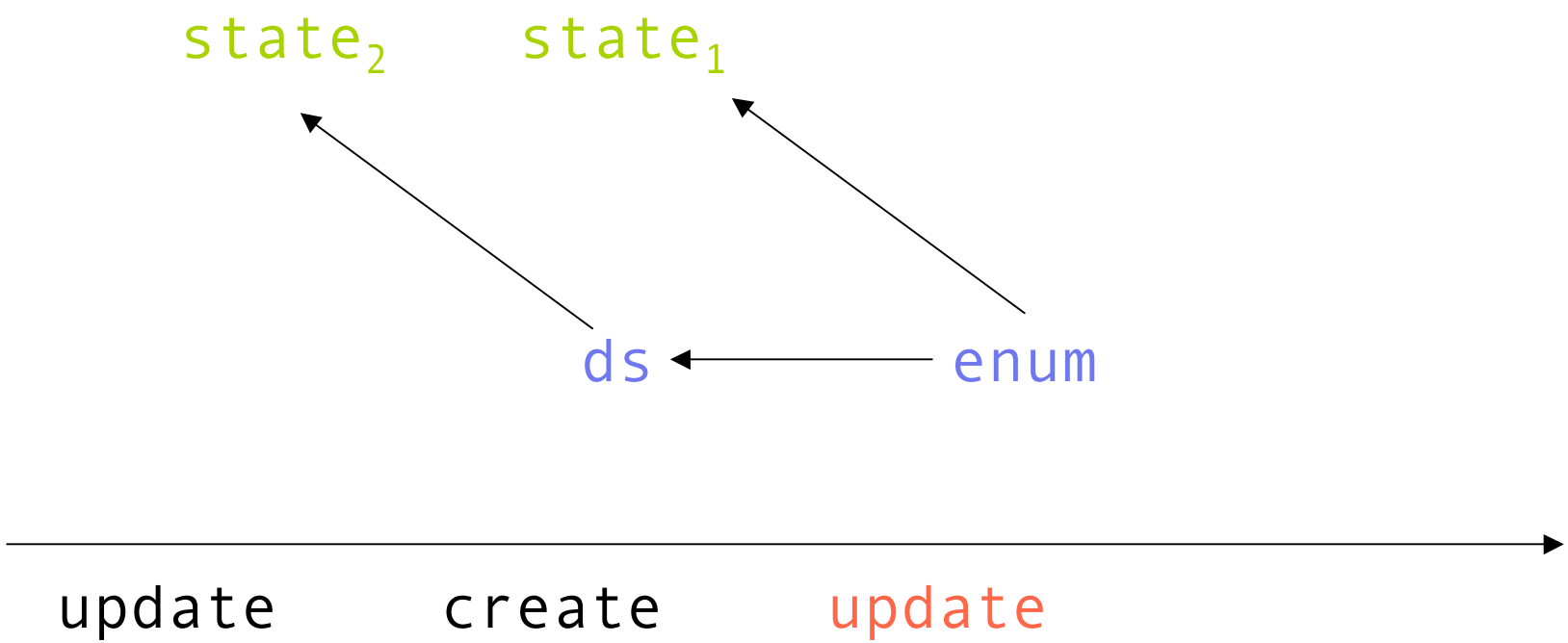
example monitored run

```
…
v.add(3);
Enumeration en = v.elements();
while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```



example monitored run

```
…
v.add(3);
Enumeration en = v.elements();
while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```



example monitored run

```
…
v.add(3);
Enumeration en = v.elements();
while(en.hasMoreElements()) {
    Integer i = (Integer)en.nextElement();
    if (i == 2)
        v.add(4);
    else
        System.out.println(i);
    }
}
```

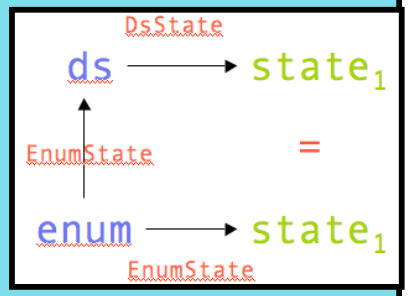
$state_2 \neq state_1$

ds ← enum

update create update next

the checker

```
aspect SafeEnum {
    private Map ds_state   = new WeakIdentityHashMap();
    private Map enum_state = new WeakIdentityHashMap();
    private Map enum_ds    = new WeakIdentityHashMap();

    private static class StateId {}

    pointcut vector_update() :
        call(* Vector.add*(..)) || call(* Vector.clear()) ||
        call(* Vector.insertElementAt(..)) || call(* Vector.remove*(..)) ||
        call(* Vector.retainAll(..)) || call(* Vector.set*(..)) && scope();

    after(Vector ds) returning(Enumeration e) :
        call(Enumeration Vector.elements()) && target(ds) {
        enum_ds.put(e,ds);
        Object s = ds_state.get(ds);
        if (s != null) enum_state.put(e,ds_state.get(ds));
    }

    before(Enumeration e):
        call(Object Enumeration.nextElement()) && target(e) {
        if (ds_state.get(enum_ds.get(e)) != enum_state.get(e))
            error("nextElement called on enumerator after update");
    }

    after(Vector ds) : vector_update() && target(ds) {
        ds_state.put(ds,new StateId());
    }
}
```



properties of Java library APIs

HashSet (Java 2 Platform SE 5.0)

http://java.sun.com/j2se/1.5.0/docs/api/

Overview Package Class Use Tree Deprecated Index Help

Java™ 2 Platform Standard Ed. 5.0

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages

java.applet

HierarchyEvent

HierarchyListener

Highlighter

HOLDING

HTML

HTML.Attrib

HTML.Tag

HTML.Unkn

HTMLDocur

HTMLDocur

HTMLEditor

HTMLEditor

HTMLEditor

HTMLEditor

HTMLEditor

HTMLEditor

java.util

Class HashSet<E>

java.lang.Object

└ java.util.AbstractCollection<E>

R_3: An collection should not be modified while it is a member of a hashset (don't change the hashcode).

E>, Collection<E>, Set<E>

e, backed by a hash table

no guarantees as to the iteration

Collection (Java 2 Platform SE 5.0)

Overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS FRAMES NO FRAMES

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Java™ 2 Platform Standard Ed. 5.0

All Classes

Packages

java.applet

CodecFactoryHelper

CodecFactoryOperation

CodecOperations

CoderMalfunctionError

CoderResult

CODESET_INCOMPATI

CodeSets

CodeSigner

CodeSource

CodingErrorAction

CollationElementIterator

CollationKey

Collator

Collection

CollectionCertStorePara

Collections

Color

ColorChooserComponer

ColorChooserUI

ColorConvertOp

java.util

Interface Collection<E>

All Superinterfaces:

Iterable<E>

All Known Subinterfaces:

BeanContext, BeanContextServices, BlockingQueue<E>, List<E>, Queue<E>, Set<E>, SortedSet<E>

All Known Implementing Classes:

AbstractCollection, AbstractList, AbstractQueue, AbstractSequentialList, AbstractSet, ArrayBlockingQueue, ArrayList, AttributeList, BeanContextServicesSupport, BeanContextSupport, ConcurrentLinkedQueue, CopyOnWriteArrayList, CopyOnWriteArraySet, DelayQueue, EnumSet, HashSet, JobStateReasons, LinkedBlockingQueue, LinkedHashSet, LinkedList, PriorityBlockingQueue, PriorityQueue, RoleList, RoleUnresolvedList, Stack, SynchronousQueue, TreeSet, Vector

don’t change hashcode

A collection should not be modified while it is member of a hash set. In other words: don’t change the collection’s hashcode during this period.

```
Set<Collection<String>> s = new HashSet();
Collection<String> c = new ArrayList();

c.add("this is ok");

s.add(c);

c.add("don't do this");

System.out.println(s.contains(c));
```

error

produces:

false

one map is needed

- recording what hashmaps a collection is stored in

$$\boxed{\texttt{Map} = \texttt{Collection} \rightarrow \texttt{HashMap-set}}$$

$$\left[\begin{array}{lcl} \texttt{coll}_1 & \longrightarrow & \{\texttt{hm}_1, \texttt{hm}_2\} \\ \texttt{coll}_2 & \longrightarrow & \{\texttt{hm}_2, \texttt{hm}_3, \texttt{hm}_4\} \\ \texttt{coll}_3 & \longrightarrow & \{\texttt{hm}_4\} \end{array}\right]$$

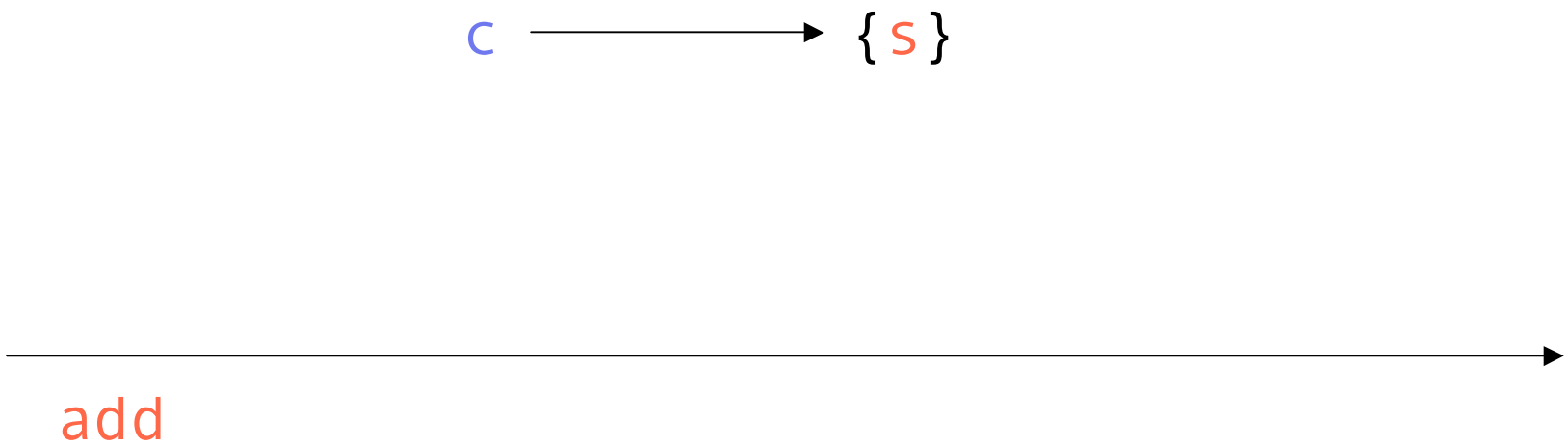
don't update collection when set is non-empty

example monitored run

```
Set<Collection<String>> s = new HashSet();
Collection<String> c = new ArrayList();

c.add("this is ok");

s.add(c);

c.add("don't do this");

System.out.println(s.contains(c));
```

example monitored run

```
Set<Collection<String>> s = new HashSet();
Collection<String> c = new ArrayList();

c.add("this is ok");

s.add(c);

c.add("don't do this");

System.out.println(s.contains(c));
```



example monitored run

```
Set<Collection<String>> s = new HashSet();
Collection<String> c = new ArrayList();

c.add("this is ok");

s.add(c);

c.add("don't do this");

System.out.println(s.contains(c));
```

$c \longrightarrow \{s\} \neq \varnothing$

add update

$c_1 \longrightarrow \{hm_1, hm_2\}$

$c_2 \longrightarrow \{hm_2, hm_3, hm_4\}$

$c_3 \longrightarrow \{hm_4\}$

```
aspect HashSetPolicy {
    pointcut addCollection(HashSet s, Collection c):
        call(* Collection.add(Object)) && target(s) && args(c);
    pointcut removeCollection(HashSet s, Collection c):
        call(* Collection.remove(Object)) && target(s) && args(c);
    pointcut modifyCollection(Collection c):
        (call(* Collection.add(..)) || …) && target(c);

    WeakIdentityHashMap hashSets = new WeakIdentityHashMap();

    after(HashSet s, Collection c) returning (): addCollection(s,c) {
        if(hashSets.get(c)==null)
            hashSets.put(c,new WeakIdentityHashSet());
        WeakIdentityHashSet sets = (WeakIdentityHashSet) hashSets.get(c);
        sets.add(s);
    }

    after(HashSet s, Collection c) returning (): removeCollection(s,c) {
        WeakIdentityHashSet sets = (WeakIdentityHashSet)hashSets.get(c);
        if(sets!=null) sets.remove(s);
    }

    before(Collection c): modifyCollection(c) {
        WeakIdentityHashSet sets = (WeakIdentityHashSet) hashSets.get(c);
        if(sets!=null)
            for (Iterator iter = sets.iterator(); iter.hasNext();)
                error("*** Modified collection "+c+" in " + iter.next());
    }
}
```

the checker

lock release policies

Within <u>one method invocation</u>, locks should be acquired and released correctly, meaning …

- release locks in same invocation they are taken
- and either one of the following policies:
  - release at most once

    `L(x) L(x) U(x) U(x)` `set`

  - release as many times as acquired (order unimportant)

    `L(x) L(x) U(x) U(x) U(x)` `bag`

  - release in reverse order

    `L(x) L(y) U(x) U(y)` `stack`

an example

```
class Lock {
  String name;

  Lock(String name) {
    this.name = name;
  }

  synchronized void lock(){…}
  synchronized void unlock(){…}

  public String toString() {
    return name;
  }
}
```

```
void start() {
    a();
}

void a() {
    l1.lock();
        l2.lock();
    l1.unlock();
        l2.unlock();
    l3.lock();
    b();
}

void b() {
    l3.unlock();
}
```

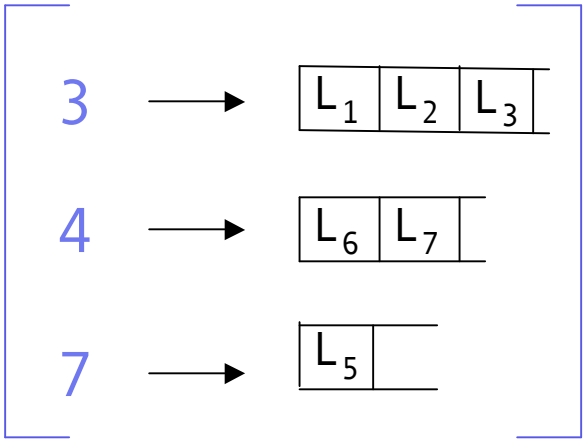
error

error

stack of lock histories

$$\text{Data} = \text{Level} \rightarrow \text{LockHist}$$

$$\cancel{\text{Data} = \text{LockHist-stack}}$$
$$\text{LockHist} = \text{Set} \mid \text{Bag} \mid \text{Stack}$$




```
public interface LockHist {
    public void lock(Lock l);
    public boolean unlock(Lock l);
    public boolean isEmpty();
    public void clear();
}
```

stack code

```
class StackHist implements LockHist {
    Stack stack = new Stack();

    public void lock(Lock l) {
        stack.push(l);
    }

    public boolean unlock(Lock l) {
        boolean success = !stack.isEmpty() && stack.peek() == l;
        if (success)
            stack.pop();
        return success;
    }

    public boolean isEmpty() {
        return stack.isEmpty();
    }

    …
}
```

computing method invocation level

ThreadLocal: this java.lang class provides thread-local variables. These variables differ from their normal counterparts in that each thread that accesses one (via its get or set method) has its own, independently initialized copy of the variable.

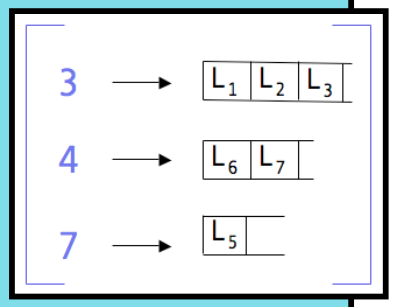
```
aspect CflowDepth {
    pointcut anyfunc() : execution(* *(..));

    static ThreadLocal cflowdepth = new ThreadLocal() {… 0 …};

    before() : anyfunc() {
        Integer prev = (Integer) cflowdepth.get();
        cflowdepth.set(new Integer(prev.intValue() + 1));
    }

    after() : anyfunc() {
        Integer prev = (Integer) cflowdepth.get();
        cflowdepth.set(new Integer(prev.intValue() - 1));
    }
}
```

initialized to zero

```
aspect LockChecker {
    public static ThreadLocal depthMap = new ThreadLocal() {… emptystack …};
    pointcut locking(Lock l)   : call(* Lock.lock()) && target(l);
    pointcut unlocking(Lock l) : call(* Lock.unlock()) && target(l);

    before(Lock l) : locking(l) {
        HashMap map = (HashMap) depthMap.get();
        Integer depth = (Integer)CflowDepth.cflowdepth.get();
        LockHist lockhist = (LockHist) map.get(depth);
        if (lockhist == null) {
            lockhist = new StackHist(); map.put(depth, lockhist);
        }
        lockhist.lock(l);
    }
    before(Lock l) : unlocking(l) {
        HashMap map = (HashMap) depthMap.get();
        Integer depth = (Integer)CflowDepth.cflowdepth.get();
        LockHist lockhist = (LockHist) map.get(depth);
        if (lockhist == null || !lockhist.unlock(l))
            error("unlock op. not preceeded by lock op." + l);
    }
    after() : CflowDepth.anyfunc() {
        HashMap map = (HashMap) depthMap.get();
        Integer depth = (Integer)CflowDepth.cflowdepth.get();
        LockHist lockHist = (LockHist) map.get(depth);
        if (lockHist != null && !lockHist.isEmpty())
            error("locks have not been released " + depth + lockHist);
        if (lockHist != null) lockHist.clear();
    }
}
```

```
declare precedence : CflowDepth, LockChecker;
```

```
new SetHist()
new BagHist()
```



the checker

end