

# Coordination aspect

- Review of AOP
- Summary of threads in Java
- COOL (COOrdination Language)
  - Design decisions
  - Implementation at Xerox PARC and for Demeter/Java

“To my taste the main characteristic of intelligent thinking is that one is willing and able to study in depth an aspect of one's subject matter in isolation, for the sake of its own consistency, all the time knowing that one is occupying oneself with only one of the aspects. ...

Quote taken from Gregor Kiczales' talk:  
[www.parc.xerox.com/aop](http://www.parc.xerox.com/aop)

- Dijkstra, *A discipline of programming*, 1976  
last chapter, **In retrospect**

A few more viewgraphs taken from Gregor Kiczales' talk  
[www.parc.xerox.com/aop](http://www.parc.xerox.com/aop)

# the goal is a clear separation of concerns

we want:

- natural decomposition
- concerns to be cleanly localized
- handling of them to be explicit
- in both design and implementation

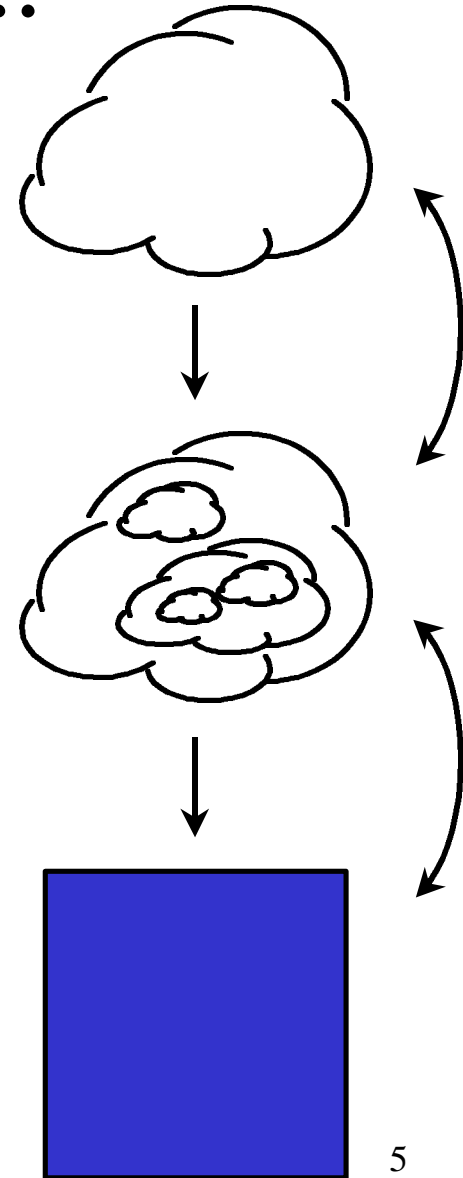


# achieving this requires...

- synergy among
  - problem structure and
  - design concepts and
  - language mechanisms

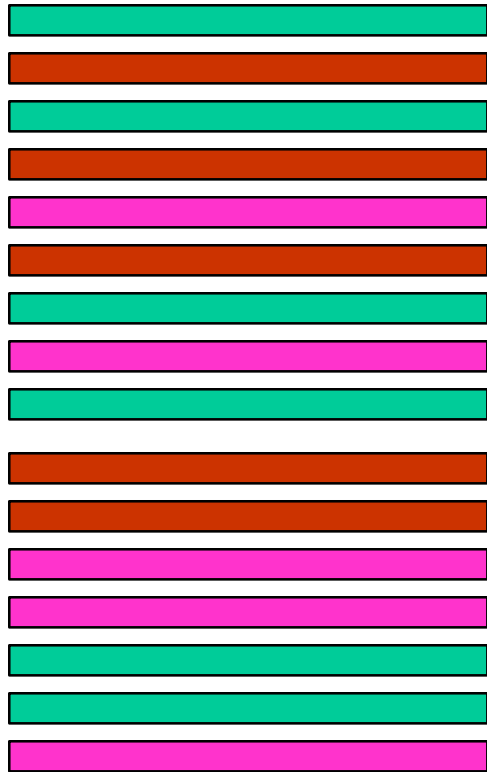
“natural design”

“the program looks like the design”

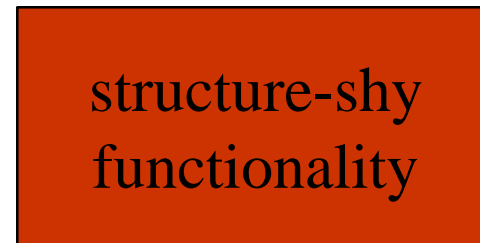


# Cross-cutting of components and aspects

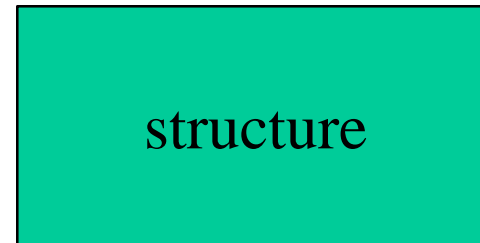
ordinary program



better program



Components

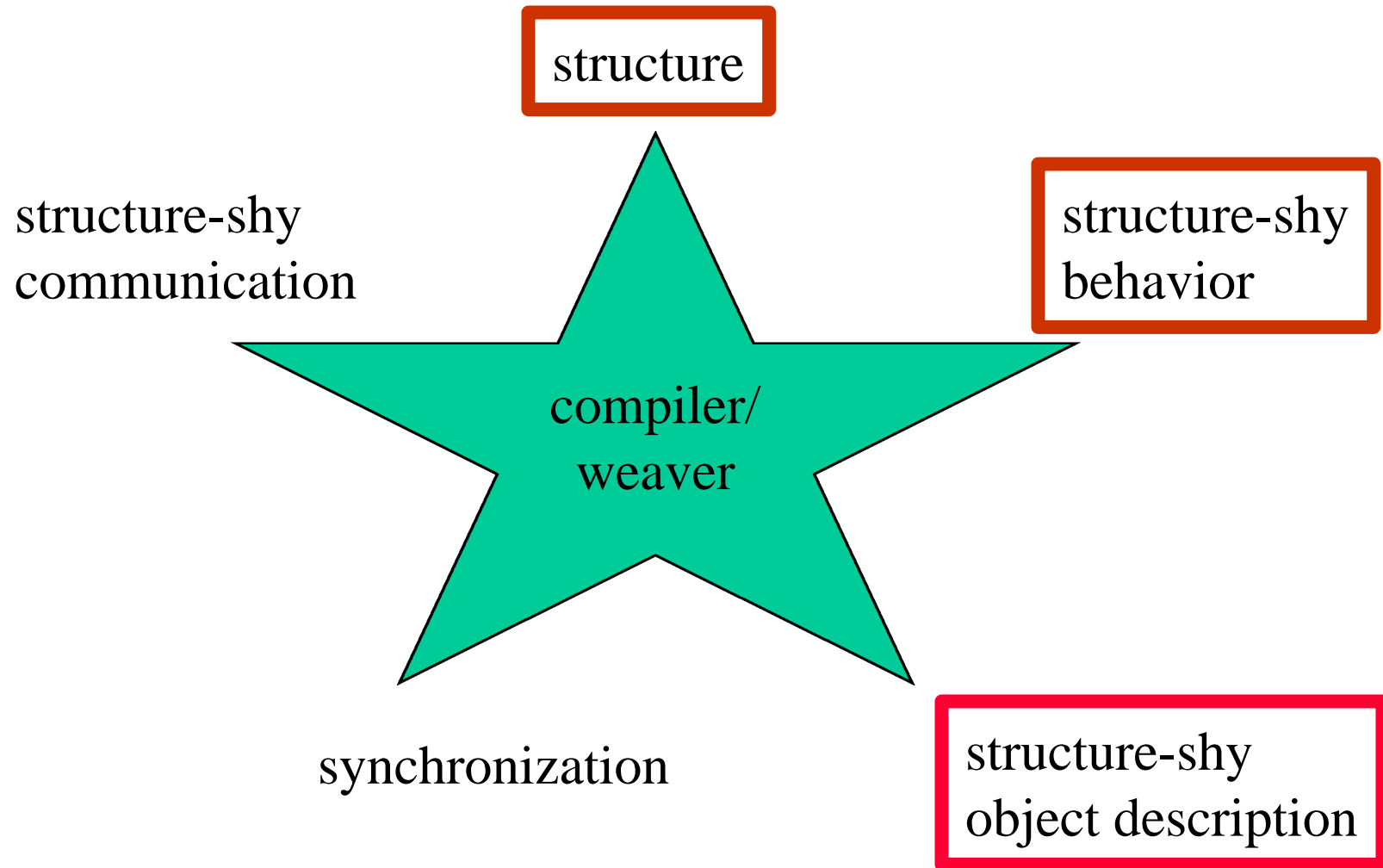


Aspect 1



Aspect 2

# Demeter



# Aspect-Oriented Programming

components and aspect descriptions

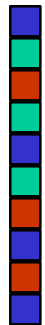
weaver  
(compile-  
time)



High-level view,  
implementation may  
be different

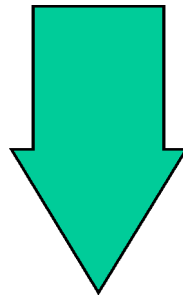


Source Code  
(tangled code)



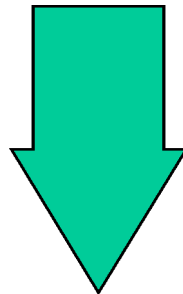
# Technology Evolution

**Object-Oriented Programming**



Law of Demeter dilemma  
Tangled structure/behavior

**Adaptive Programming**



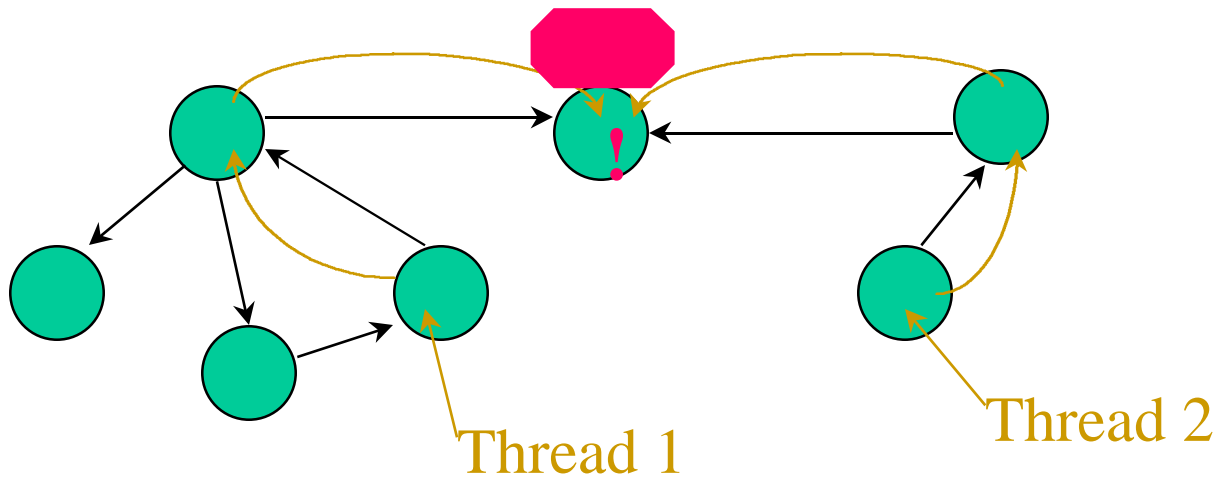
Other tangled code

**Aspect-Oriented Programming**

# Components/Aspects of Demeter

- Functionality (structure-shy)
  - Traversal (Traversal Strategies)
  - Functionality Modification (Visitors)
- Structure (UML class diagrams)
- Description (annotated UML class diagrams, class dictionaries)
- Synchronization

# Threads



# Coordination aspect

- Put coordination code about thread synchronization in one place.
- Threads are synchronized through methods.
- Method synchronization
  - Exclusion sets
  - Method managers

# Java Threads

- Thread class in Standard Java libraries
- `Thread worker = new Thread( )`
- `start` method spawns a new thread of control based on Thread object. `start` invokes the threads `run` method: active thread

# Java Threads

- synchronized method: locks object. A thread invoking a synchronized method on the same object must wait until lock released. Mutual exclusion of two threads.

```
class Account {  
    synchronized double getBalance() { return balance;}  
    synchronized void deposit(double a) { balance += a;}  
}
```

# Java Threads

- synchronized statements  
    synchronized ( expr )  
        statement
- lock an object without invoking a synchronized method
- `expr` must produce an object to lock

# Java Threads

- communication between threads with `wait` and `notify` (defined in class `Object`).
- cliché:

```
synchronized
```

```
void doWhenCondition() {  
    while (!condition) wait();  
    do_it(); }  
}
```

# Java Threads

- notify: after change of data that some other thread is waiting on

```
synchronized void changeCondition() {  
    // change some value  
    notifyall();  
    // wakes all waiting threads  
}
```

# Problem with synchronization code: it is tangled with component code

```
class BoundedBuffer {  
    Object[] array;  
    int putPtr = 0, takePtr = 0;  
    int usedSlots = 0;  
    BoundedBuffer(int capacity){  
        array = new Object[capacity];  
    }  
}
```

# Tangling

```
synchronized void put(Object o) {  
    while (usedSlots == array.length) {  
        try { wait(); }  
        catch (InterruptedException e) {};  
    }  
    array[putPtr] = o;  
    putPtr = (putPtr + 1 ) % array.length;  
    if (usedSlots==0) notifyall();  
    usedSlots++;  
    // if (usedSlots++==0) notifyall();  
}
```

# Solution: tease apart basics and synchronization

- write core behavior of buffer
- write coordinator which deals with synchronization
- use weaver which combines them together
- simpler code
- replace `synchronized`, `wait`, `notify` and `notifyall` by coordinators

Using Demeter/Java, \*.beh file

## With coordinator: basics

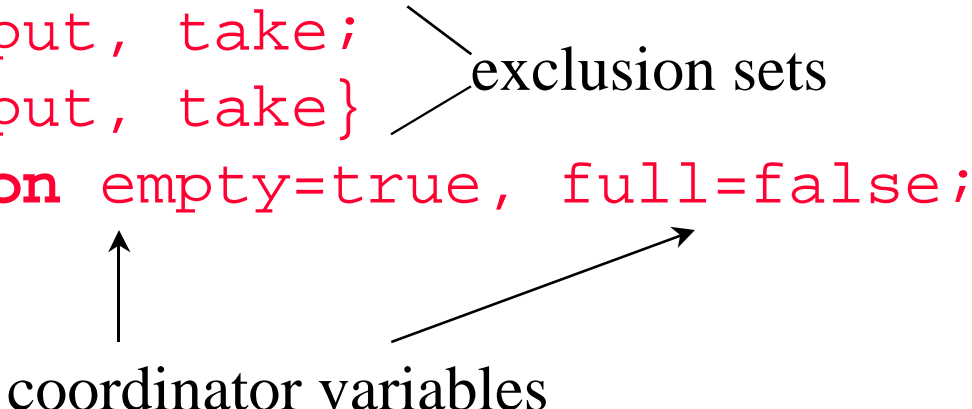
```
BoundedBuffer {
public void put (Object o) (@
    array[putPtr] = o;
    putPtr = (putPtr+1)%array.length;
    usedSlots++; @)
public Object take() (@
    Object old = array[takePtr];
    array[takePtr] = null;
    takePtr = (takePtr+1)%array.length;
    usedSlots--;
    return old; @)
```

Using Demeter/COOL, put into \*.cool file

# Coordinator

```
coordinator BoundedBuffer {  
  selfex put, take;  
  mutex {put, take} } exclusion sets  
  condition empty=true, full=false;  
}
```

↑  
coordinator variables



# Coordinator

method managers with *requires* clauses and *entry/exit* clauses

```
put requires (!full) {
    on exit {empty=false;
            if (usedSlots==array.length)
                full=true; }}
take requires (!empty) {
    on exit {full=false;
            if (usedSlots==0)
                empty=true; }}
}
```

# exclusion sets

- `selfex A.f, B.g;`
  - only one thread can call a selfex method
  - `A.f` and `B.g` may run simultaneously.
- `mutex {g, h, i} mutex {f, k, l}`
  - if a thread calls a method in a mutex set, no other thread may call a method in the same mutex set.

# Multi-class coordination supported

```
coordinator A, B {  
  selfex A.put, B.take;  
  mutex {B.put, A.take}  
  condition empty=true, full=false;  
  ...
```

# Design decisions behind COOL

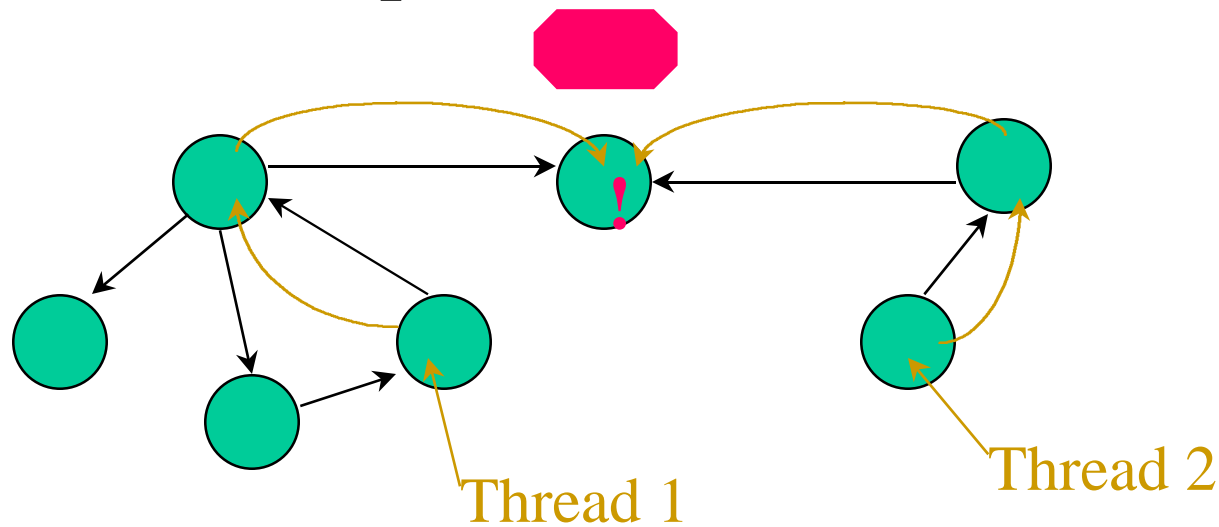
- The smallest unit of synchronization is the method.
- The provider of a service defines the synchronization (monitor approach).
- Coordination is contained within one coordinator.
- Association from object to coordinator is static.

# Design decisions behind COOL

- Deals with thread synchronization within each execution space. No distributed synchronization.
- Coordinators can access the objects' state, but they can only modify their own state. Synchronization does not “disturb” objects. Currently a design rule not checked by implementation.

# COOL

- Provides means for dealing with mutual exclusion of threads, synchronization state, guarded suspension and notification



coordinator

# COOL

- Identifies “good” abstractions for coordinating the execution of OO programs
  - coordination, not modification of the objects
  - mutual exclusion: sets of methods
  - preconditions on methods
  - coordination state (history-sensitive schemes)
  - state transitions on coordination

## plain Java

```
public class Shape {
    protected double x_ = 0.0;
    protected double y_ = 0.0;
    protected double width_ = 0.0;
    protected double height_ = 0.0;

    double x() { return x_(); }
    double y() { return y_(); }
    double width(){
        return width_();
    }
    double height(){
        return height_();
    }
    void adjustLocation() {
        x_ = longCalculation1();
        y_ = longCalculation2();
    }
    void adjustDimensions() {
        width_ = longCalculation3();
        height_ = longCalculation4();
    }
}
```

11/12/98

## COOL Shape

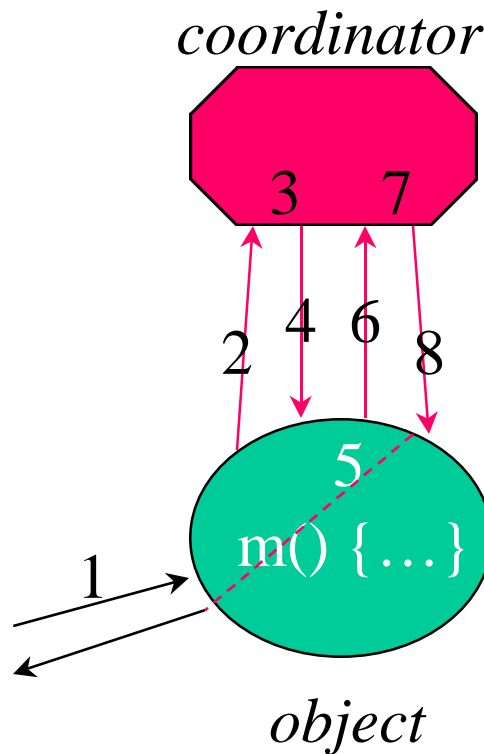
```
coordinator Shape {
    selfex {adjustLocation,
            adjustDimensions}
    mutex {adjustLocation,x}
    mutex {adjustLocation,y}
    mutex {adjustDimensions,
            width}
    mutex {adjustDimensions,
            height}
}
```

AOO / Demeter / NU

30

# Programming with COOL

Protocol object/coordinator:



1: method invocation

2: request presented to the coordinator

3: coordinator checks synchronization state, eventually suspending thread; when thread can proceed, coordinator performs “on\_entry” actions

4: request proceeds to the object

5: method execution

6: return is presented to coordinator

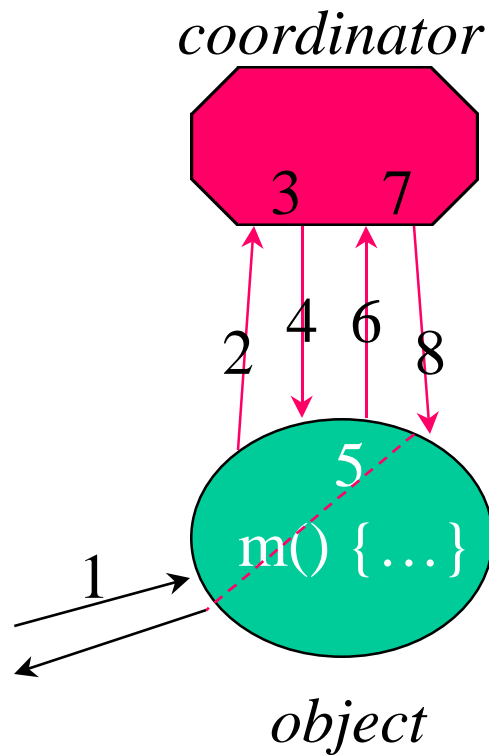
7: coordinator performs “on\_exit” actions

8: method returns

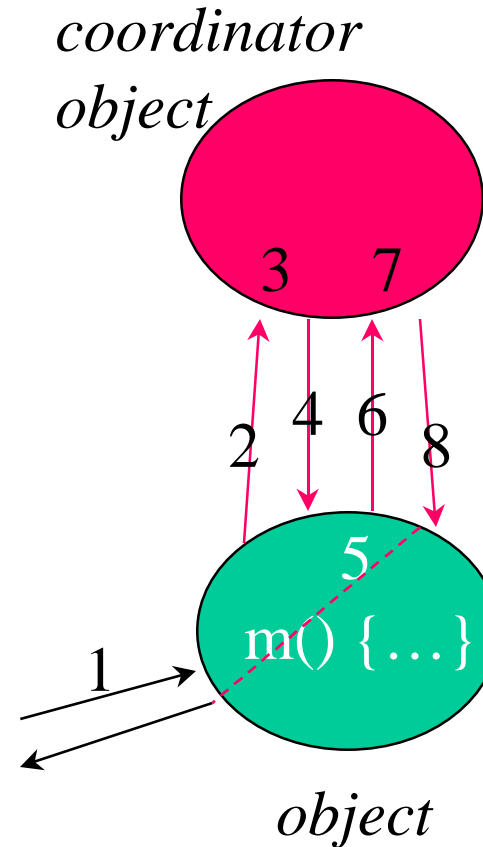
# COOL View of Classes

- Stronger visibility:
  - coordinator can access:
    - all methods and variables of its classes, independent of access control
    - all non-private methods and variables of their superclasses
- Limited actions:
  - only read variables, not modify them
  - only coordinate methods, not invoke them

# Programming with *COOL*



Semantics



Implementation

# COOL

```
public class BoundedBuffer {
    private Object array[];
    private int putPtr = 0, takePtr = 0;
    private int usedSlots = 0;

    public BoundedBuffer(int capcty){
        array = new Object[capcty];
    }

    public void put(Object o) {
        array[putPtr] = o;
        putPtr = (putPtr+1)%array.length;
        usedSlots++;
    }

    public Object take() {
        Object old = array[takePtr];
        array[takePtr] = null;
        takePtr = (takePtr+1)%array.length;
        usedSlots--;
        return old;
    }
}
```

```
coordinator BoundedBuffer {
    selfex {put, take}
    mutex {put, take};
    boolean full=(@ false @),
           empty=(@ true @);
    put requires (@ !full @) {
        on exit (@
            empty = false;
            if (usedSlots==array.length)
                full = true;
        @)
    }
    take requires (!empty){
        on exit (@
            full = false;
            if (usedSlots == 0)
                empty = true;
        @)
    }
}
```

# Implementing COOL

```

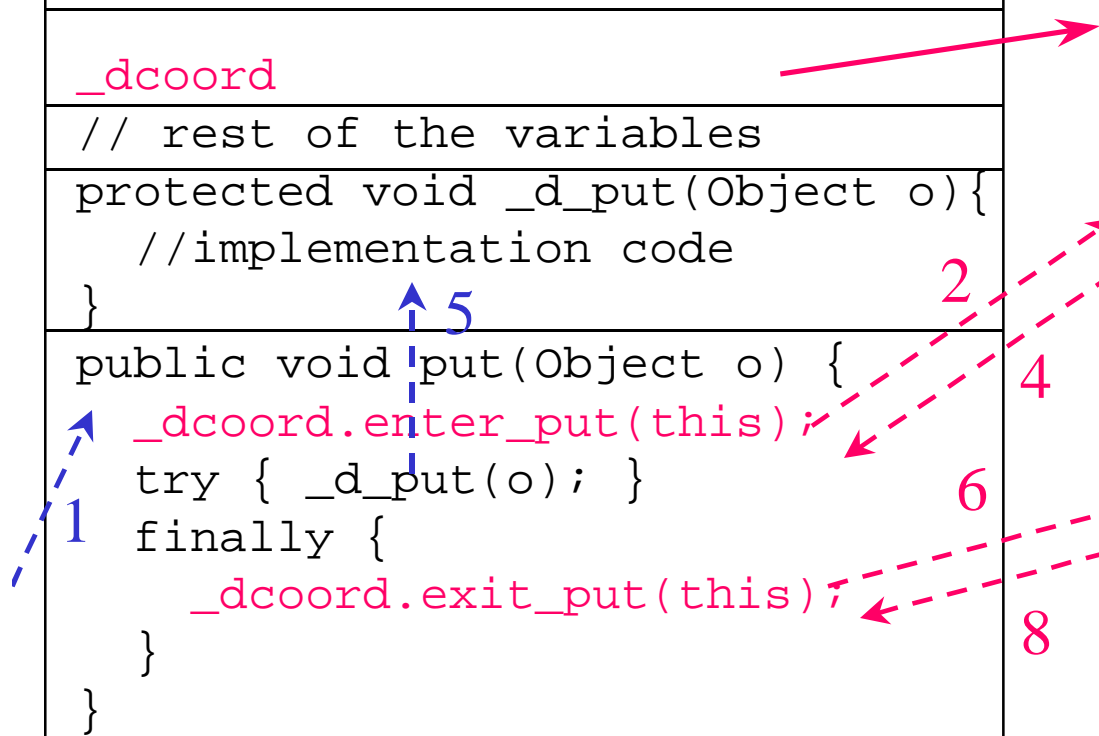
BoundedBuffer
// rest of the variables
protected void _d_put(Object o){
    //implementation code
}
public void put(Object o) {
    1  _dcoord.enter_put(this);
    try { _d_put(o); }
    finally {
        _dcoord.exit_put(this);
    }
}
protected Object _d_take() {
    //implementation code
}
public Object take() {
    //similar to put
}
11/12/98

```

```

BoundedBufferCoord
// variable next page
synchronized void
enter_put(BoundedBuffer o){
    // ...
}
synchronized void
exit_put(BoundedBuffer o) {
    // ...
}
synchronized void
enter_take(BoundedBuffer o){
    // ...
}
synchronized void
exit_take(BoundedBuffer o) {
    // ...
}

```



## Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
              take.isBusyByOtherThread() ||
              full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots()==o._dget_size()) full=true;
        notifyAll();
    }
}
```

# Implementing COOL

```
class BoundedBufferCoord {
```

```
    MethState put = new MethState();
```

```
    MethState take = new MethState();
```

```
    boolean empty = true, full = false;
```

```
    public synchronized void enter_put(BoundedBuffer o) {
```

```
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
```

```
            try { wait(); }
```

```
            catch (InterruptedException e) {}
```

```
        }
```

```
        put.in();
```

```
    }
```

```
    public synchronized void exit_put(BoundedBuffer o) {
```

```
        put.out();
```

```
        empty = false;
```

```
        if (o._dget_usedlots() == o._dget_size()) full = true;
```

```
        notifyAll();
```

```
    }
```

```
11/12/98
```

```
AOO / Demeter / NU
```

```
37
```

```
...
```

*to keep track of method  
execution state*

# One class to support COOL

```
class MethodState {
    int depth = 0;
    Vector t1 = new Vector();
    final public boolean isBusyByOther() {
        if (depth > 0 &&
            !t1.contains(Thread.currentThread()))
            return true;
        else return false;
    }
    final public void in() { depth++;
        t1.addElement(Thread.currentThread());
    }
    final public void out() { depth--;
...}
```

# Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    coordination vars

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots()==o._dget_size()) full=true;
        notifyAll();
    }
}
```

# Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {} conditions for waiting
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots() == o._dget_size()) full = true;
        notifyAll();
    }
}
```

...

# Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();           update method state
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots() == o._dget_size()) full = true;
        notifyAll();
    }
}
```

## Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots() == o._dget_size()) full = true;
        notifyAll();
    }
}
```

*update method state*

# Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots()==o._dget_size()) full=true;
        notifyAll();
    }
}
```

*on\_exit statements*

11/12/98

AOO / Demeter / NU

43

....

# Implementing COOL

```
class BoundedBufferCoord {
    MethState put = new MethState();
    MethState take = new MethState();
    boolean empty = true, full = false;

    public synchronized void enter_put(BoundedBuffer o) {
        while (put.isBusyByOtherThread() ||
               take.isBusyByOtherThread() ||
               full) {
            try { wait(); }
            catch (InterruptedException e) {}
        }
        put.in();
    }

    public synchronized void exit_put(BoundedBuffer o) {
        put.out();
        empty = false;
        if (o._dget_usedlots()==o._dget_size()) full=true;
        notifyAll();
    }
}
```

*notify state change*

...

# Acknowledgments

- Many of the viewgraphs prepared by Crista Lopes for her Ph.D. work supported by Xerox PARC.
- Implementation of COOL for Demeter/Java by Josh Marshall. Integration into Demeter/Java with Doug Orleans.
- ECOOP '94 paper on synchronization patterns by Lopes/Lieberherr.